Minicurso – NFe no Delphi XE 2 – Parte 2: Certificados digitais e seu uso

Clube Delphi

PostgreSQL
Conexão via ODBC e ZeosLib

Aplicando Threads
Divida para conquistar

Ano 12 :: Edição 139 :: R$ 11,90

DevMedia

EXPRESSÕES REGULARES

Regex na prática com Delphi XE2

Boas Práticas
Padrões de projeto: Decorator e Singleton

Firebird
Desmistificando o uso do Execute Block

ISSN 1517990-7

Somos tão apaixonados por tecnologia que o nome da empresa

diz tudo.

Porta 80 é o melhor que a Internet pode oferecer para sua empresa.

Já completamos 8 anos e estamos a caminho dos 80, junto com nossos clientes.

Adoramos tecnologia.
Somos uma equipe composta de gente que entende e gosta do que faz,

assim como você.

Estrutura

100% NACIONAL. Servidores de primeira linha, links de alta capacidade.

Suporte diferenciado

Treinamos nossa equipe para fazer mais e melhor. Muito além do esperado.

Serviços

Oferecemos a tecnologia mais moderna, serviços diferenciados e antenados com as suas necessidades.

1-to-1

Conhecemos nossos clientes. Atendemos cada necessidade de forma única. Conheça!

Porta 80
WEB HOSTING

Hospedagem | Cloud Computing | Dedicados | VoIP | Ecommerce | Aplicações | Streaming | Email corporativo

porta80.com.br | comercial@porta80.com.br | twitter.com/porta80

SP 4063-8616 | RJ 4063-5092 | MG 4063-8120 | DF 4063-7486

Sumário

Delphi

Easy — **06 – PostgreSQL, Conexão via ODBC e ZeosLib** [*Giuliano Scombatti Pinto*]

Delphi — **12 – Expressões Regulares** [*Giuliano Scombatti Pinto*]

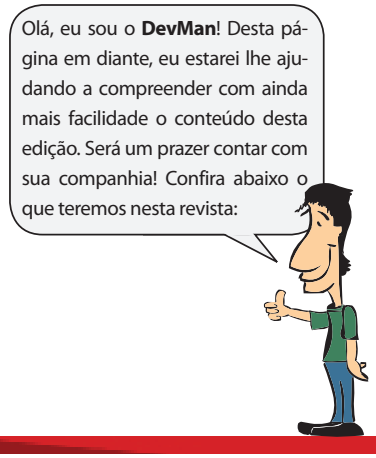
Delphi — **20 – Aplicando Threads** [*Giuliano Scombatti Pinto*]

Minicurso — **27 – NF2 no Delphi XE 2 – Parte 2** [*Giuliano Scombatti Pinto*]

Boas Práticas — **34 – Desmistificando o uso do Execute Block** [*Daniel Sobrinho Laporte*]

Boas Práticas

Boas Práticas — **42 – Padrões de Projeto: Decorator e Singleton** [*Renato Groffe*]



Dê seu feedback sobre esta edição!

A ClubeDelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre esta edição, artigo por artigo, através do link:

www.devmedia.com.br/clubedelphi/feedback

Clube Delphi

Ano 12 - 139ª Edição 2012 - ISSN 1517990-7 Impresso no Brasil

Corpo Editorial

Editor Geral

Paulo Quicoli
pauloquicoli@devmedia.com.br

Editor Técnico

Daniel Sobrinho Laporte
daniel.laporte@gmail.com

Equipe Editorial

Vitor Luiz Rubio
vitorrubio@gmail.com

Rafael Stavarengo
faelsta@gmail.com

Guinther Pauli
guintherpauli@gmail.com

Jornalista Responsável

Kaline Dolabella - JP24185

Atendimento ao Leitor

A DevMedia conta com um departamento exclusivo para o atendimento ao leitor. Se você tiver algum problema no recebimento do seu exemplar ou precisar de algum esclarecimento sobre assinaturas, exemplares anteriores, endereço de bancas de jornal, entre outros, entre em contato com:

www.devmedia.com.br/mancad
(21) 3382-5038

Publicidade

Para informações sobre veiculação de anúncio na revista ou no site e para fechar parcerias ou ações específicas de marketing com a DevMedia, entre em contato com:

publicidade@devmedia.com.br

Distribuição

FC Comercial e Distribuidora S.A
Rua Teodoro da Silva, 907
Grajaú - RJ - 206563-900

EDITORIAL

O ser humano por natureza possui o medo do desconhecido. Daquilo que não conhecemos, ficamos distantes. Na programação às vezes acontece o mesmo, alguns recursos não são utilizados por parecerem obscuros demais. As expressões regulares caem nessa situação. Podemos citar aqui um milhão de vantagens para o seu uso, mas quando vemos somente algo como: ^\d{2}-\d{4}-\d{4}$, saímos correndo. Para acabar com isso, o Giuliano traz um artigo que explica e fundamenta o uso das expressões regulares no Delphi XE2, mostrando como seu uso pode acabar com várias "gambiarras" que fazemos quando o assunto é tratamento de texto.

Reza a lenda que o Firebird 3 está chegando, porém, o que adianta termos essa versão se continuamos a usá-lo como se fosse a versão 1.5? O Daniel Laporte apresenta o recurso Execute Block do Firebird, disponível a partir da versão 2. Com ele podemos executar blocos de comandos completos, como se fosse uma Stored Procedure, só que de forma dinâmica. Ao entender seu funcionamento vocês verão como isso é vantajoso. Falando no Firebird 3, que será multithread, temos um outro artigo do Giuliano que trata exatamente disso, de threads. Como você pode utilizar o processamento dividido para gerar mais performance ao seu aplicativo. Nesse artigo um exemplo bem interessante é criado, não perca.

Ainda temos a continuação do minicurso sobre NFe e Delphi XE 2, que trata dos certificados digitais. Neste artigo você saberá como assinar digitalmente as notas fiscais que uma vez assinadas, precisam ser validadas antes de seu envio à SEFAZ.

E para quem está começando com Delphi e gosta de experimentar bancos de dados diferentes, temos uma introdução ao uso do PostgreSQL com ODBC e ZeosLib.

Finalizando, para quem já tem experiência na orientação a objetos, seguem mais dois padrões de projeto, o Decorator e o Singleton, apresentados por nosso amigo Renato.

Boa leitura!

Paulo Quicoli
pauloquicoli@gmail.com
Editor Geral
quicoli.wordpress.com
twitter.com/pauloquicoli

MVP

Assine agora e tenha acesso a todo o conteúdo da DevMedia: **www.devmedia.com.br/mvp**

DevMedia

Fale com o Editor!

É muito importante para a equipe saber o que você está achando da revista: que tipo de artigo você gostaria de ler, que artigo você mais gostou e qual artigo você menos gostou. Fique a vontade para entrar em contato com os editores e dar a sua sugestão!
Se você estiver interessado em publicar um artigo na revista ou no site ClubeDelphi, entre em contato com os editores, informando o título e mini-resumo do tema que você gostaria de publicar:

Paulo Quicoli - Editor da Revista
pauloquicoli@gmail.com

Sistema de créditos DevMedia

Conteúdo **completo** ao seu alcance

Com o sistema de créditos DevMedia você pode comprar individualmente mais de 4.000 vídeo aulas, 8000 artigos, além de palestras e vídeos com formações completas.

TODAS as vídeo aulas que você sempre quis ver a partir de

R$ 1,20

ECONOMIA: você não precisa assinar 8 produtos diferentes para ter acesso ao conteúdo completo do site!

Saiba mais sobre o Sistema de Créditos!

www.devmedia.com.br/creditos

DevMedia

Conexão PostgreSQL
Acesso via ODBC e ZeosLib

A integração de aplicações e bancos de dados é um padrão adotado e muito necessário na maioria das vezes nos projetos de sistemas. O Sistema Gerenciador de Banco de Dados (SGBD) PostgreSQL, também conhecido como Postgres, surgiu por volta de 1986 nos Estados Unidos, mais especificamente na Universidade de Berkeley - CA e sempre possuiu código-fonte aberto, sendo assim, ele pode ser estudado, alterado, utilizado e distribuído por qualquer pessoa e para qualquer fim, comercial, pessoal, etc. É um SGBD do tipo objeto-relacional (ORD - Object-relational Database) de nível corporativo, sendo também multiplataforma, possuindo diversas versões para Windows, Linux, Solaris, Mac OS X, entre outros Sistemas Operacionais de plataformas 32 e 64 bits. Ele também conta com grande parte dos tipos de dados de acordo com o ISO SQL: 1999 como Numeric, Integer, Varchar, Timestamp, Boolean, etc, tendo suporte a junções, gatilhos, procedures, chaves estrangeiras, Views e campos binários que podem ser utilizados para o armazenamento de vídeos, sons ou imagens.

O Postgres é um SGBD elaborado, preparado e otimizado para atender grandes corporações e entidades que necessitem ou trabalhem com um alto volume de informações, fornecendo assim uma base consistente para o armazenamento dos dados.

No que diz respeito a suas funcionalidades, o Postgres possui registrador de transações sequencial, ferramentas de recuperação, controle de concorrência multiversionado, replicação assíncrona, planejador de consultas, ferramentas para cópias de segurança, etc. Também é um SGBD extremamente escalável no que diz respeito ao número de usuários concorrentes e na quantidade de informações que são gerenciadas. Por ser um projeto aberto, o Postgres também depende de uma comunidade de desenvolvedores ao redor do planeta que possam atuar na sua evolução, suporte e na identificação de possíveis erros. Vale à pena lembrar também que existe uma equipe que mantém o núcleo do projeto para gerenciar e dar sequência ao mesmo.

O Postgres conta com várias distribuições efetuadas por terceiros que possuem um determinado conjunto de ferramentas adicionais, como é o caso do Postgres Plus, além de contar com diversas distribuições Live CD baseadas em Ubuntu, Fedora, entre outros, que permitem a utilização do mesmo sem que exista a necessidade da instalação, onde o Software é encontrado pré-configurado e com grande parte das ferramentas.

A arquitetura do Postgres, assim como em outros servidores de bancos de dados, é do tipo cliente-servidor. Ele é constituído de um programa servidor de banco de dados que é denominado Postmaster. Sendo assim, gerencia as conexões das aplicações cliente, bem como comanda e efetua as ações solicitadas nos arquivos do banco de dados. Em contrapartida, as aplicações clientes que se conectarão ao servidor podem utilizar ferramentas de comandos, aplicações gráficas, entre outros. O servidor também aceita várias conexões simultâneas, onde a cada nova conexão é criado um novo processo que opera de maneira independente do processo original do Postmaster. Entretanto o Postgres pode ser executado em modo monousuário através da aplicação que leva seu nome (postgres).

Resumo DevMan

De que trata o artigo:

Este artigo aborda uma iniciação ao sistema gerenciador de banco de dados Postgres, visando a integração de uma aplicação desenvolvida no Delphi XE 2 com uma base de dados criada neste SGBD, utilizando o padrão de conexões ODBC para aplicações FireMonkey ou VCL, e também, o componente ZeosLib para acesso nativo aos dados, que também pode ser utilizado em ambas as plataformas.

Em que situação o tema é útil:

Permitir que desenvolvedores possam integrar seus softwares desenvolvidos no Delphi XE 2 na plataforma FireMonkey ou VCL com o SGBD Postgres de uma maneira fácil e rápida. Demonstrar a praticidade de trabalhar com diferentes tipos de bancos de dados de forma simplificada.

PostgreSQL:

O SGBD Postgres é um software livre e de código-fonte aberto que surgiu a mais de 25 anos e que vem sendo atualizado e mantido por diversas comunidades de desenvolvedores. Além da possibilidade de ser utilizado em aplicações desenvolvidas no Delphi XE 2, o Postgres também é muito utilizado na Internet, em sistemas de comércio eletrônico, devido a várias de suas características como a robustez. Contudo, o Postgres é otimizado para ser adotado em aplicações complexas, onde o volume de dados é muito alto e existem diversas informações críticas. Este artigo demonstra uma breve iniciação ao banco de dados mencionado, demonstrando a integração dos bancos de dados com a plataforma FireMonkey, utilizando para isso o padrão ODBC, ou a plataforma VCL, através da biblioteca de acesso nativo ZeosLib.

no Delphi XE 2

Giuliano Scombatti Pinto

O Postgres também conta com uma aplicação através de prompt de comando para iniciar, pausar, reiniciar ou parar o servidor, que é denominada pg_ctl. Através dela também é possível obter informações de seu status.

Para concluir, não existe um tamanho máximo estipulado para as bases de dados criadas no Postgres, o tamanho máximo que uma tabela pode ter é 32 Tb, o que comprova a capacidade de lidar com um enorme volume de informações.

Instalação, configuração e acesso

Por ser um sistema livre, o Postgres pode ser obtido facilmente direto do site da comunidade do mesmo (ver seção links), em forma de instalador executável ou através de arquivo compactado, inclusive com seu código-fonte disponível. Para os exemplos demonstrados neste artigo, a versão adotada é a 8.4 de 32 bits. Durante a instalação, são solicitadas diversas informações importantes para sua configuração, como é o caso da senha do usuário padrão (postgres) e da porta de acesso. Depois de instalado, vale à pena lembrar que o aplicativo servidor do Postgres deve ser iniciado para que as conexões possam ser executadas, o que normalmente é feito automaticamente. Também é interessante evidenciar que ao concluir a instalação, é criado um grupo de programas no menu Iniciar que possui diversas ferramentas, onde é possível iniciar e parar o servidor e etc. O Postgres também conta com uma aplicação de terminal interativa denominada psql, onde o desenvolvedor pode efetuar consultas e alterações através de comandos SQL, além de criar e apagar bases de dados, a partir de aplicações gráficas, para possibilitar e facilitar seu acesso , como é o caso do PgAdmin, entre outras ferramentas.

Nele também é possível efetuar transações através de um terminal ou do PgAdmin. A transação é um conceito muito importante nos sistemas gerenciadores de banco de dados, pois permite que vários comandos distintos sejam executados ou não, de uma única vez, em outras palavras, ou todas as ações são satisfatoriamente efetuadas ou nenhuma é realizada. O comando utilizado para iniciar uma transação é BEGIN enquanto para a conclusão é COMMIT.

O Postgres também possui um arquivo onde podem ser efetuadas as configurações do servidor, como o número máximo de usuários que podem se conectar. Este arquivo é denominado postgresql.conf e quando alterado, o servidor do Postgres deve ser reiniciado.

No que diz respeito a sua utilização em uma rede onde o servidor é uma máquina remota, caso as aplicações não consigam identificar o mesmo, o desenvolvedor pode também criar as variáveis de ambiente PGHOST para informar o nome do servidor e PGPORT para informar a porta.

Nota do DevMan

O ISO SQL: 1999, também denominado SQL3, foi um padrão estabelecido em uma das revisões da linguagem SQL (Structured Query Language), a linguagem padrão adotada pelos Sistemas Gerenciadores de Bancos de Dados para a manipulação e pesquisa das informações contidas em uma base de dados.

O Postgres também é permite diversos comandos que podem ser facilmente utilizados através do prompt , simplesmente acessando ou adicionando a pasta de binários ao Path do Windows, lembrando que a mesma se encontra no diretório padrão: "C:\Arquivos de programas\PostgreSQL\8.4\bin". A **Tabela 1** demonstra alguns comandos que podem ser utilizados no Postgres através do prompt, para o correto funcionamento, devem ser fornecidos os parâmetros necessários para a utilização dos mesmos, e caso existam dúvidas sobre os comandos, basta adicionar o parâmetro "--help" para obter ajuda.

Para dar início à integração da aplicação Delphi com o Postgres, o primeiro passo é criar um banco de dados. Para tanto, basta abrir o programa PgAdmin através do grupo de programas do Postgres localizado no menu Iniciar. A interface sugere a opção de Novo Banco de Dados, onde deve ser definido seu nome e dono, que no caso deste projeto foram definidos como "DADOS" e "postgres" respectivamente. Também é possível definir outras configurações como os usuários ou grupos de usuários que possuem acesso ao banco, além de modelos, variáveis, entre outros. Com o banco de dados criado, já torna-se possível o desenvolvimento e execução dos códigos SQL. Para criar campos com autoincremento, como é o caso da chave-primária, o Postgres possui algumas particularidades, utilizando uma combinação de comandos. A **Tabela 2** descreve os recursos que podem ser utilizados para a utilização de autoincremento em campos de uma tabela.

Sendo assim, antes de criar a tabela, deve-se utilizar o comando "CREATE SEQUENCE CLIENTES_CODIGO;" para criar o Sequence utilizado para a geração da chave-primária da tabela de clientes. A seguir é criada a tabela utilizada. O código a seguir cria a tabela CLIENTES.

```
CREATE TABLE CLIENTES (
 CODIGO INTEGER DEFAULT NEXTVAL('CLIENTES_CODIGO') NOT NULL,
 NOME VARCHAR(50),
 ENDERECO VARCHAR(50),
 CIDADE VARCHAR(50),
 TELEFONE VARCHAR(15),
 CONSTRAINT PKY_CLIENTES PRIMARY KEY (CODIGO)
);
```

O campo denominado "CODIGO" recebe os comandos necessários para utilização do autoincremento, bem como o nome do sequence que deve ser usado. Sequence é um objeto do banco de dados que tem a funcionalidade de representar um sequenciamento de números. Através dele mantemos um número, que é incrementado posteriormente. Este incremento pode ser realizado como mostrado no código anterior, assim que um novo registro for inserido na tabela Clientes, o banco de dados gera o próximo número da sequência e este é retornado ao campo codigo. Esse sequenciamento ocorre fora do contexto de uma transação, ou seja, uma vez gerado o próximo número, não é possível reaver o anterior através de uma operação de RollBack. Uma operação de RollBack consegue voltar o estado de objetos que estão dentro de uma transação e se essa transação ainda não foi finalizada com o comando COMMIT.

Delphi XE 2 e Postgres com ODBC

Existem várias maneiras de conectar uma aplicação Delphi XE 2 com o Postgres, utilizando os componentes do próprio Delphi ou componentes de terceiros, através de soluções pagas ou gratuitas. Nesta versão do Delphi, o dbExpress possui um novo driver ODBC que permite que aplicações desenvolvidas na plataforma VCL ou FireMonkey possam se conectar facilmente a qualquer banco de dados que possua um driver neste mesmo padrão para o Windows. Sendo assim, para integrar o dbExpress com o Postgres, além do SGBD instalado e configurado, deve-se instalar o driver ODBC que pode ser facilmente obtido pela própria comunidade de desenvolvimento do Postgres (ver seção links). O ODBC (Open Data Base Connectivity) é um recurso padronizado independente de Sistema Operacional e linguagem de programação que é constituído de interfaces padrões que permitem a integração das mais variadas aplicações com diversos Sistemas Gerenciadores de Bancos de Dados, uma vez que o driver para o SGBD esteja disponível, permitindo assim o acesso aos arquivos de dados para os programas. Sendo assim, o ODBC utiliza uma camada intermediária para a realização da conexão com o banco de dados, o que torna esse tipo de conexão um pouco mais lenta que um acesso nativo. Contudo, é importante salientar que existem versões deste driver para ambientes Windows 32 e 64 bits, portanto, deve-se utilizar o driver de acordo com o ambiente atual. Com o driver ODBC instalado, basta realizar a configuração da Fonte de Dados para que ela fique disponível as aplicações. Para tanto, a configuração deve ser feita em Ferramentas Administrativas>Fontes de Dados ODBC dentro do painel de controle. Na aba "Fonte de dados de usuário", deve-se clicar no botão Adicionar, selecionar a opção "PostgreSQL Unicode" e após concluir, efetuar a configuração dos dados do banco. A **Figura 1** ilustra a janela de configuração ODBC para o PostgreSQL.

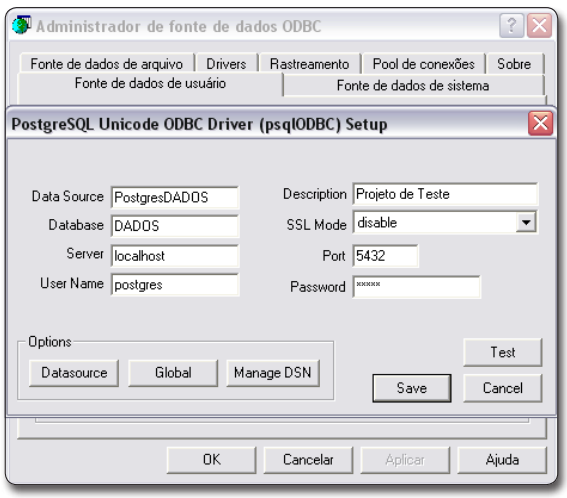


Figura 1. Janela de configuração do Postgres através de ODBC

Comando ou Parâmetro	Descrição
created [nome bd]	Tem o mesmo efeito de CREATE DATABASE, permitindo assim a criação de uma base de dados.
dropdb [nome bd]	Tem o mesmo efeito de DROP DATABASE, o que faz com que o banco de dados informado seja apagado.
createuser [nome]	É utilizado para a criação de uma conta de usuário.
dropuser [nome]	É utilizado para apagar uma conta de usuário.
pg_config	Exibe informações do Postgres.
pg_dump	Permite a gravação de um banco de dados do Postgres para um arquivo ou script etc. Este comando é muito útil no que diz respeito à criação de backups (cópias de segurança das informações).
pg_restore	Permite a restauração de um banco de dados através do arquivo gerado pelo pg_dump.
--help	Parâmetro que pode ser adicionado aos comandos para obter ajuda e diversas informações sobre os mesmos.

Tabela 1. Alguns comandos uteis que podem ser utilizados através do prompt

Comando	Descrição
Sequence	É utilizado para que o Postgres possa gerar números sequenciais.
Nextval	É utilizado para a obtenção do próximo valor de acordo com uma Sequence.
Default value	É utilizado para a definição de um valor padrão que deve ser usado caso não seja informado nenhum valor quando for executado um comando Insert.

Tabela 2. Comandos que podem ser utilizados para o autoincremento de campos no Posgtres

Com a configuração da fonte de dados efetuada, cria-se um novo projeto "FireMonkey HD Application" para dar início a integração da aplicação Delphi XE 2 com o Postgres. Sendo assim, a conexão é realizada por meio de um componente SQLConnection. A **Figura 2** demonstra a configuração das propriedades necessárias para a integração com o ODBC através do dbExpress. A propriedade IsolationLevel merece atenção porque ela configura o nível de isolamento das transações. Existem basicamente dois níveis principais de isolamento em todo banco de dados relacional: Read Commited e Dirty Read. Basicamente quando uma transação opera no regime Read Commited, dentro daquela transação só são vistos dados já finalizados (que sofreram um COMMIT). Já o modelo Dirty Read permite que uma transação veja informações, dados que ainda não foram sofreram um COMMIT. Mas qual o impacto e diferença entre esses isolamentos? Para compreender vamos ver uma situação:

Um usuário alterou o registro nº 2 de uma tabela e ainda NÃO deu o Commit. Imagine agora que em outra máquina outro usuário que quer ler os dados do registro 2. Se o modo de operação da transação é Read Commited, isso significa que o segundo usuário vai enxergar os dados antigos do registro e não vai ver a alteração em andamento. Caso a opção selecionada seja Dirty Read, o usuário poderá ler os dados que ainda não sofreram COMMIT.

O modo dirty read não é recomendado porque pode levar a inconsistências, por exemplo, o que aconteceria se o primeiro usuário não confirmasse sua alteração? Todos os dados obtidos pelo segundo usuário não seriam mais verdadeiros.

No exemplo é utilizada a instrução SQL "SELECT * FROM CLIENTES" na propriedade ConnectionText de um SimpleDataSet. Como de costume, as ligações dos componentes são realizadas por meio de um DataSource através da propriedade DataSource, da mesma que é realizada com um driver nativo. Com a estrutura básica de conexão ao banco de dados e a tabela criada, são inseridos cinco componentes Edit para a interligação com os campos da tabela Clientes através de LiveBindings. Com um dos Edits selecionado, é selecionada a propriedade "Link to DB Field" para que o Delphi crie automaticamente os objetos necessários para a utilização do recurso LiveBindings e, evidentemente, interligue o Edit com o campo da tabela. O mesmo é realizado com o objeto BindNavigator, porém, na propriedade BindScope, onde é selecionado o Bind ScopeDB para permitir a navegação e edição dos dados da tabela. Por último, o código que realiza a conexão e abertura do componente para posterior exibição dos dados, como a seguir.

```
procedure TForm1.FormCreate(Sender: TObject);
begin
  SQLConnection1.Open;
  SimpleDataSet1.Open;
end;

procedure TForm1.FormDestroy(Sender: TObject);
begin
  SimpleDataSet1.Close;
  SQLConnection1.Close;
end;
```

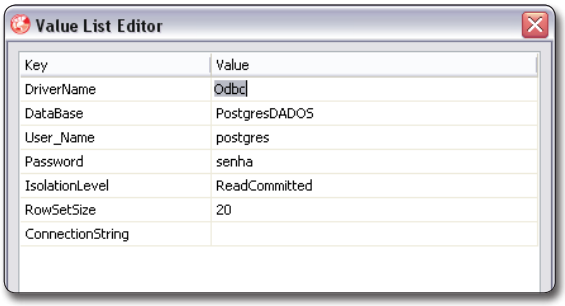
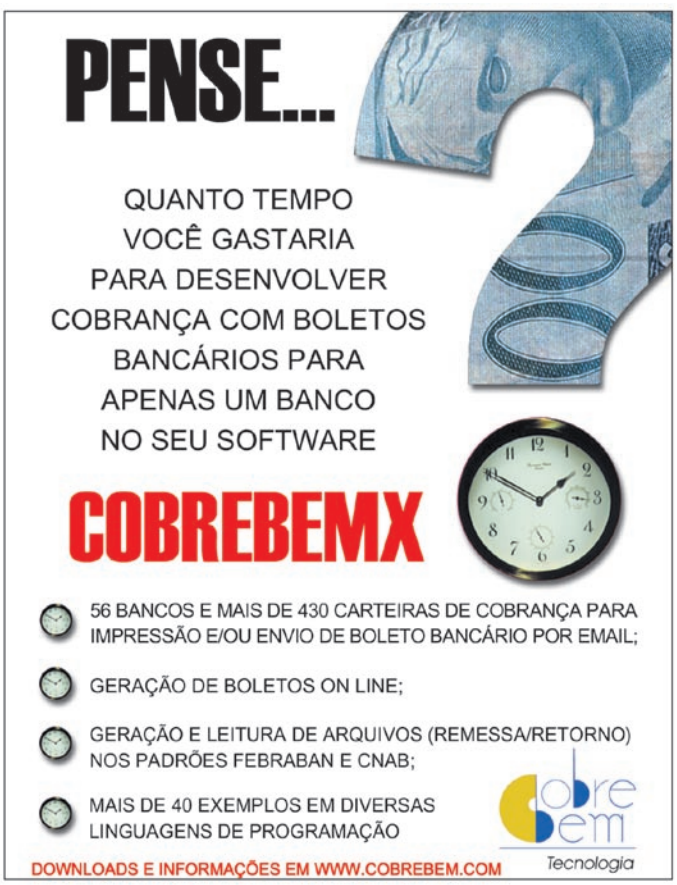


Figura 2. Configuração das subpropriedades do Params

Definidos todos os objetos e suas propriedades para este exemplo de integração com Postgres através de ODBC, são codificadas as ações para gravação das informações na tabela de clientes, através dos eventos AfterDelete e AfterPost do mesmo. O método ApplyUpdates envia para o banco de dados as alterações realizadas no SimpleDataSet.

```
procedure TForm1.SimpleDataSet1AfterDelete(DataSet: TDataSet);
begin
  SimpleDataSet1.ApplyUpdates(0);
end;

procedure TForm1.SimpleDataSet1AfterPost(DataSet: TDataSet);
begin
  SimpleDataSet1.ApplyUpdates(0);
end;
```

Para concluir este exemplo com FireMonkey, o interessante é atribuir um visual diferenciado à aplicação utilizando um estilo ou tema, através do objeto TStyleBook. Para tanto, é adicionado um objeto do tipo StyleBook ao formulário carregando um estilo pré-definido (duplo clique) localizado por padrão no diretório "C:\Documents and Settings\All Users\Documentos\RAD Studio\9.0\Styles". Aplicando o estilo após a seleção a propriedade StyleBook do formulário é preenchida com o componente. A **Figura 3** demonstra a aparência final da aplicação FireMonkey integrada ao Postgres com ODBC utilizando um estilo visual.

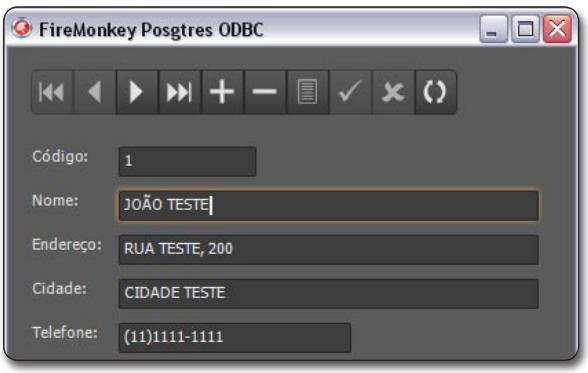


Figura 3. Aparência final da aplicação ODBC com FireMonkey e Postgres

Nota do DevMan

O LiveBindings, adicionado na versão XE 2 do Delphi, é um recurso baseado em expressões relacionais para interligar os objetos através de suas propriedades. Este recurso pode ser utilizado em aplicações na plataforma FireMonkey ou VCL e podem ser unidirecionais ou bidirecionais. Utiliza um conceito presente em outras tecnologias como .net por exemplo, aonde existe o conceito de Binding. Nele qualquer componente pode ser ligado a uma fonte de dados, que pode até mesmo ser uma coleção de outros objetos.

Delphi XE 2 e Postgres com ZeosLib

Para a conexão nativa entre uma aplicação Delphi XE 2 com o Postgres, a biblioteca conhecida como ZeosLib, ZeosDBO ou simplesmente ZEOS pode ser usada. O ZeosLib é uma suíte gratuita de código-fonte aberto criada a vários anos, constituída de diversos objetos que permitem a conexão nativa com vários Sistemas Gerenciadores de Banco de Dados, dentre eles o Postgres, disponibilizando assim a manipulação das informações das bases de dados de uma maneira rápida. O componente ZeosLib pode ser utilizado na plataforma VCL e também funciona no FireMonkey. As versões mais antigas deste componente podem não funcionar corretamente com a versão XE 2 do Delphi, portanto, o ideal é obter versões mais recentes, como é o caso da 7.0.0 utilizada neste exemplo (ver seção de links). Considerando que o componente ZeosLib esteja devidamente instalado e configurado, é criado um novo projeto VCL Forms Application. Os passos são os mesmos de outras suítes. O componente de conexão (ZConnection) para a conexão nativa com o Postgres, ZTable para a manipulação dos registros da tabela clientes, e ZSequence para definir a Sequence que será utilizada, ambos da paleta Zeos Access. As propriedades que devem ser configuradas no ZConnection são demonstradas na **Tabela 3**.

Além das propriedades informadas na **Tabela 3**, também se deve definir em Properties do ZConnectioin os valores "codepage=UTF8" e "client_encoding=UTF8" para que o ZeosLib possa reconhecer acentos e caracteres especiais. Contudo, ao tentar efetuar a conexão com o banco de dados pode surgir uma mensagem de erro informando que as bibliotecas libpq81.dll ou libpq.dll não foram encontradas. Este erro é muito comum e pode ser facilmente corrigido copiando as bibliotecas (arquivos *.dll) do diretório "C:\Arquivos de programas\PostgreSQL\8.4\bin" para a pasta de sistema. No ZTable é informado em sua propriedade TableName a tabela "public.clientes" e ligando-o ao DataSource como de costume. Para ativar a Sequence deve-se interligar o ZTable ao ZSequence através da sua propriedade Sequence e na propriedade SequenceField definir o campo "codigo". No componente ZSequence deve-se informar o nome da Sequence criada (public.clientes_codigo), lembrando que a Sequence é utilizada para a função de autoincremento do campo de código da tabela clientes. Basta um DBNavigator para que o layout esteja finalizado.

Assim como no exemplo anterior, os componentes devem ser ativados/desativados através dos eventos OnShow e OnClose, como é mostrado a seguir.

```
procedure TForm1.FormShow(Sender: TObject);
begin
  ZConnection1.Connect;
  ZTable1.Open;
end;

procedure TForm1.FormClose(Sender: TObject; var Action: TCloseAction);
begin
  ZTable1.Close;
  ZConnection1.Disconnect;
end;
```

Assim como outras suítes de acessado a dados, a conexão é realizada de maneira simples através de simples comandos de abertura. Contudo, o exemplo ilustra como melhorar ao visual da VCL Application através dos recursos do VCL Styles. Este recurso permite adicionar a todos os formulários e componentes do sistema, diversos skins (temas) para modificar o visual. Um estilo pode ser aplicado facilmente através da opção Project>Options e na opção denominada Appearance, dentro de Application, é informado o estilo desejado. No exemplo é adotada a skin Aqua Graphite. A **Figura 4** demonstra o visual final do exemplo VCL e Postgres utilizando ZeosLib e VCL Styles.

Conclusão

O Postgres é um Sistema Gerenciador de Banco de Dados sólido e de código-fonte livre, criado e mantido há muito tempo por sua comunidade de desenvolvedores, que pode ser utilizado e distribuído gratuitamente. É utilizado em ambientes corporativos onde o volume de dados normalmente é alto, bem como na

Propriedade	Valor	Descrição
Protocol	postgresql-8	Definição do SGBD e versão utilizada.
HostName	localhost	Definição do nome do servidor, no caso a máquina local.
Port	5432	Definição da porta de conexão.
User	postgres	Nome do usuário.
Password	senha	Senha do usuário informado.
Database	DADOS	Nome da base de dados.
Transact IsolationLevel	tiReadCommitted	Isolation Level da Transação.

Tabela 3. Propriedades definidas do objeto ZConnection1 para conexão com Posgtres

Internet, permitindo a manipulação de diversas informações de maneira consistente. Assim como a maioria dos SGBDs disponíveis, o Postgres possui suporte aos mais diversos recursos da linguagem SQL, bem como transações, Views, Procedures, sendo ainda multiusuário. Outra grande vantagem do Postgres é o fato do mesmo estar disponível para várias plataformas, como é o caso do Windows e do Linux, em 32 ou 64 bits.

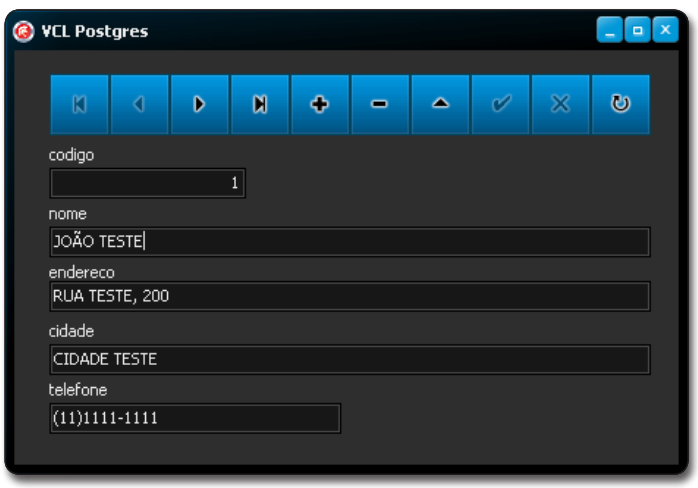


Figura 4. Visual final do exemplo com Postgres, ZeosLib e VCL Styles

A interligação do Postgres com aplicações desenvolvidas em Delphi XE 2, seja na plataforma VCL ou FireMonkey, pode ser realizada de diversas maneiras, utilizando recursos do próprio ambiente de desenvolvimento, de terceiros, gratuitos ou pagos. Como visto neste artigo, uma das maneiras mais fáceis e rápidas de integrar o Postgres com os projetos é através do padrão ODBC, uma vez que o Postgres possui o driver necessário para tanto, porém, este tipo de conexão é um pouco mais lenta. Também foi demonstrado como integrá-lo usando a biblioteca ZeosLib, um recurso também gratuito e de código-fonte aberto que pode ser obtido facilmente, podendo ser adicionado ao ambiente de desenvolvimento do Delphi e que permite o acesso nativo ao arquivo de dados, não somente para integração com o Postgres, mas também para outros bancos de dados, como é o caso do Firebird e do MySQL. Para concluir, também foram apresentadas duas alternativas, uma em FireMonkey e outra em VCL, para dar um visual mais personalizado e moderno à aplicação.

Giuliano Scombatti Pinto
giuliano@sygnux.com.br
É Analista Desenvolvedor da Sygnux Software (www.sygnux.com.br) localizada em Monte Alto/SP, atuando também com desenvolvimento Web e Android. Formado pela Fatec/TQ, começou a desenvolver com Delphi em 2003. Atualmente trabalha com Delphi, Java, PHP e Flash/Flex. Professor de Informática de Projetos Sociais da Prefeitura Municipal de Monte Alto/SP.

Download do PostgreSQL
http://www.postgresql.org.br/downloads

Download do driver ODBC para PostgreSQL
http://ftp.postgresql.org/pub/odbc/versions/msi/

Download do ZeosLib
http://zeos.firmos.at/portal.php

DevSpace do Autor
http://www.devmedia.com.br/space.asp?id=283274

Dê seu feedback sobre esta edição!

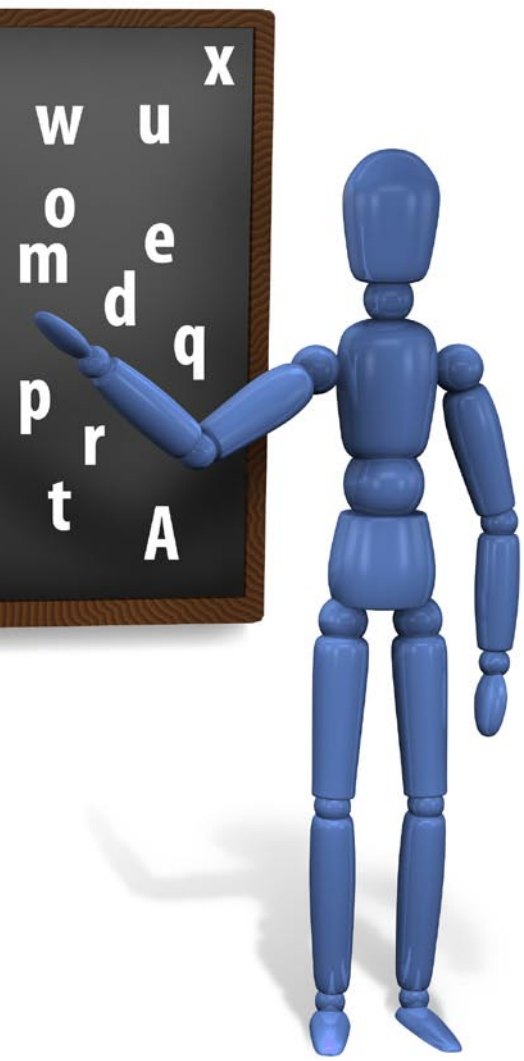
A Clubedelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre este artigo, através do link:

www.devmedia.com.br/clubedelphi/feedback

Desmistificando Expr

Colocando em prática com o Delphi XE2



Resumo DevMan

De que trata o artigo:

Este artigo aborda uma introdução ao recurso de Expressões Regulares, demonstrando também como utilizar o mesmo nas aplicações criadas no Delphi XE 2, para efetuar validações de acordo com expressões, pesquisar palavras de uma maneira rápida e simplificada, e a substituir termos em textos.

Em que situação o tema é útil:

Facilitar e aumentar a produtividade no desenvolvimento de aplicações que necessitem validar as informações presentes nos campos preenchidos pelos usuários, apresentando também uma alternativa para ser utilizada em aplicações que trabalham com edição de texto, facilitando a busca e substituição de palavras.

Expressões Regulares:

Na área da informática existem diversos padrões e conceitos que podem ser utilizados para facilitar e agilizar determinadas tarefas que fazem parte do cotidiano dos desenvolvedores. O recurso de Expressões Regulares é baseado na utilização de um padrão pré-definido para o reconhecimento de um determinado trecho em um texto ou campo, bem como para a validação do conteúdo do mesmo de uma maneira objetiva, onde o desenvolver apenas necessita ter uma familiarização com os símbolos ou objetos utilizados para a elaboração do padrão e dos objetos e eventos necessários para a implementação do mesmo através do Delphi. Neste artigo, primeiramente é apresentada uma introdução aos símbolos e caracteres utilizados na elaboração das expressões. Posteriormente, são desenvolvidos diversos exemplos em Delphi para validar informações, pesquisar e substituir palavras em textos de uma maneira bem simplificada, utilizando o recurso de RegEx.

Desde muito tempo atrás, a utilização de Expressões Regulares, também conhecidas como RegEx (Regular Expression), é muito aplicada e presente no cenário da informática, nos mais diversos softwares e ferramentas presentes no dia-a-dia das pessoas, uma vez que podem aumentar a produtividade e facilitar a vida de desenvolvedores. Muitas vezes o usuário pode ter contato com este recurso e nem se dar conta, principalmente ao utilizar uma aplicação de busca ou através de um website na Internet, quando efetua o preenchimento de seus dados nos formulários, onde campos como o email são checados.

A expressão regular é constituída de um conjunto de caracteres ou elementos combinados, também conhecidos como metacaracteres, que formam um padrão pré-estabelecido que pode ser aplicado para a obtenção de determinada informação de acordo com o mesmo, onde o termo "casamento" também pode ser adotado, que em outras palavras significa que a informação está de acordo com o padrão apresentado.Sendo assim, através da utilização de uma expressão regular é possível efetuar buscas mais objetivas, substituir palavras em um texto, efetuar uma validação das informações fornecidas, entre outros recursos.

O conceito de Expressões Regulares não é uma particularidade de determinada linguagem ou ambiente operacional, mas sim um recurso que pode ser adotado nas

essões Regulares

Giuliano Scombatti Pinto

mais diversas plataformas e ferramentas para desenvolvimento, como é o caso do Delphi XE 2, onde tal recurso foi incorporado a partir da versão XE.

A expressão regular normalmente é formada por vários processos. Um deles muito utilizado é a alternância, que consiste na utilização de barra vertical "|" para separar os valores que podem ser assumidos, agindo como a função de "ou", em outras palavras, quando é informado o valor "verdadeiro|verdadeira", os valores "verdadeiro" ou "verdadeira" podem ser considerados válidos. Outro processo também muito adotado éa utilização de agrupamento, que é obtido através do uso dos parênteses "( )". Sendo assim, ao informar a expressão "verdadeir(o|a)s", os valores "verdadeiros" ou "verdadeiras" podem ser considerados válidos. Para concluir a estrutura básica que pode compor uma expressão regular, outro processo que pode ser aplicado é o da repetição, indicando assim o número de vezes que determinado padrão pré-estabelecido pode existir. Os símbolos *, + e ? são os identificadores mais utilizados e comuns para determinar a quantificação. O símbolo * determina que pode existir nenhuma, uma ou várias ocorrências do elemento que o precede, ou seja, se a expressão "go*l" for fornecida, os valores gl, gol, gooooool, etc, podem ser admitidos. Já o símbolo + também implica que determinado elemento precedente pode ter várias ocorrências, porém deve haver pelo menos uma, sendo assim, se o elemento for a expressão "go+l", os valores admitidos serão gol, gool, goooool, etc. Com relação ao símbolo ?, o mesmo indica que pode haver apenas uma ou nenhuma ocorrência do termo precedente, ou seja, se utilizada a expressão "go?l", os valores gl e gol seriam válidos. Evidentemente todos estes processos podem ser combinados entre si para que o objetivo possa ser atingido e assim formarem expressões mais complexas.

Como é possível observar, as Expressões Regulares possuem vários símbolos que devem ser utilizados de acordo com a necessidade em questão.

Expressões regulares e seus caracteres e o Delphi XE 2

Como mencionado anteriormente, o recurso de Expressões Regulares foi incluído no Delphi a partir da versão XE, porém existem diversos componentes de terceiros que podem ser instalados e utilizados em versões anteriores da ferramenta. O objeto mais comum responsável pelo recurso de Expressões Regulares no Delphi XE 2 é denominado TRegEx e está presente na Unit System.RegularExpressions. O TRegEx é um registro que utiliza funções que pertencem á classe TPerlRegEx, que por sua vez faz uso da PCRE (Perl Compatible Regular Expressions), uma biblioteca de código fonte aberto desenvolvida na linguagem C que foi inspirada na semântica e sintaxe da linguagem de programação Perl, cujo o objetivo é oferecer diversas funções para a implementação do recurso de Expressões Regulares em vários ambientes.

A **Tabela 1** demonstra os métodos que podem ser utilizados pelo objeto TRegEx no Delphi XE 2.

Como mencionado anteriormente, existem diversos caracteres e símbolos que podem ser combinados para serem utilizados na elaboração de uma expressão regular, sendo a familiarização e conhecimento dos mesmos essencial para a elaboração das expressões. A **Tabela 2** demonstra alguns dos principais caracteres que podem ser utilizados.

Sendo assim, a expressão regular pode ser elaborada e armazenada em uma variável do tipo String e carregada pela instância do TRegEx para efetuar o processo desejado do recurso de RegEx.

Método	Descrição
Create	Utilizado para criar uma instância do registro TRegEx.
Match	É utilizado para efetuar a busca da sequência de entrada definida para a primeira ocorrência de uma expressão regular especificada, retornando um valor do tipo TMatch.
IsMatch	Uma função que retorna um valor booleano indicando se foi encontrada ou não uma correspondência em uma sequência de entrada de acordo com a expressão regular fornecida.
TMatch	Um registro utilizado para armazenar as informações de retorno para o primeiro padrão de uma sequência.
TMatchCollection	Utilizado para armazenar várias instâncias do TMatch.
Replace	Utilizado para substituir as ocorrências de determinados caracteres por uma sequência de caracteres de substituição informado de acordo com o padrão de uma maneira muito simples e ágil.

Tabela 1. Métodos que podem ser utilizados com o TRegEx

Caractere	Descrição
{}	As chaves são utilizadas para agrupar as expressões
[]	Os colchetes são utilizados para indicar os caracteres ou faixa de caracteres que podem ser utilizados. Sendo assim, ao utilizar [abc], compreende-se que são permitidos os caracteres a, b ou c.
[^]	O circunflexo presente entre os colchetes é utilizado para indicar os caracteres ou faixa de caracteres que não devem ser utilizados, sendo assim, ao utilizar [^abc], os caracteres a, b e c não são permitidos
[-]	Ao ser utilizado entre colchetes, a função do hífen é especificar a faixa de caracteres permitidos, sendo assim, ao utilizar [a-d], os caracteres a, b, c e d podem ser utilizados.
\	A barra invertida antes de um caractere curinga é utilizada para que o interpretador descarte sua função curinga, ou em outras palavras, \^ faz com que seja admitido o caractere ^ e não sua função de início de cadeia ou de valores não permitidos.
.	O ponto é um curinga utilizado para indicar que qualquer caractere pode ser usado.
^	Quando o circunflexo é utilizado no início de uma cadeia, indica o início de uma linha
*	O asterisco é utilizado como curinga para indicar que nenhum, um ou mais caracteres possam ser utilizados.
+	O sinal de adição indica que é permitida a utilização de um ou mais caracteres, evidentemente de acordo com sua posição.
$	O cifrão é utilizado para indicar o final de uma linha.
%	O sinal de porcentagem indica que o início da linha coincide com o início da cadeia.
?	Utilizado para indicar qualquer caractere único.
|	Utilizada para indicar uma alternativa. Sendo assim, carro|moto indica que podem ser utilizadas as palavras carro ou moto.
@	Utilizado para obrigar a ocorrência de um caractere na expressão.

Tabela 2. Caracteres utilizados em Expressões Regulares

Caractere	Descrição
[:blank:]	Indica a tabulação e espaço, equivalente a [\t] em ASCII.
[:digit:]	Corresponde a utilização de dígitos, equivalente a [0-9] em ASCII.
[:alpha:]	Indica a utilização de caracteres alfabéticos, equivalente a [A-Za-z] em ASCII.
[:alnum:]	Indica os caracteres alfanuméricos, correspondentes a [A-Za-z0-9] em ASCII.
[:cntrl:]	Corresponde aos caracteres de controle, representados por [\x00-\x1F\x7F] em ASCII.
[:upper:]	Utilização dos caracteres em maiúsculo, equivalente a [A-Z] em ASCII.
[:lower:]	Utilização dos caracteres em minúsculo, equivalente a [a-z] em ASCII.
[:graph:]	Representa os caracteres visíveis, correspondentes a [\x21-x7E] em ASCII.
[:punct:]	Indica os caracteres referentes a pontuação, o equivalente a [-!"#$%&'()*+,./:;<=>?@[\\\]_`{|}~] em ASCII.
[:print:]	Corresponde aos caracteres visíveis e espaços, o mesmo que [\x20-x7E] em ASCII.
[:space:]	Indica os caracteres de espaço em branco, equivalentes a [\t\r\n\v\f] em ASCII.
[:xdigit:]	Utilizado para dígitos hexadecimais, representados por [A-Fa-f0-9] em ASCII.

Tabela 3. Métodos padrões de Acesso para grupos de caracteres

Com a utilização de Expressões Regulares em diversos ambientes, foi estabelecida uma para padronizar e facilitar algumas categorias de caracteres. Existe um método padrão que pode ser utilizado para definir grupos de caracteres. A **Tabela 3** demonstra alguns métodos padrões de acesso para os grupos comentados anteriormente.

Uma outra regra adotada no uso de Expressões Regulares é a utilização da Sequência de Escape. Tal sequência consiste na combinação de uma contrabarra "\" com dígitos ou uma letra, para ser utilizada na representação de caracteres de controle. Ela também permite o envio de caracteres de controle para a impressora. Sendo assim, existem algumas sequências de controle que foram adotadas na linguagem Perl e posteriormente também em outras diversas plataformas devido a sua utilidade e facilidade.A **Tabela 4** demonstra várias sequências de escape muito úteis para a aplicação em uma expressão regular.

Para concluir, os outros caracteres que são muito utilizados na composição de expressões regularessão os destinados a repetição. A **Tabela 5** demonstra os caracteres utilizados para este fim.

Tendo todas as tabelas anteriores em mãos, já é possível iniciar o estabelecimento de algumas expressões regulares. Por exemplo, para validar se uma determinada string está dentro do padrão de um CPF, a expressão ^\d{3}\.\d{3}\.\d{3}\-?\d{2}$ pode ser utilizada. Como pode ser observado nela o símbolo ^ representa o início da expressão, o \d{3} define que o valor informado deve ser uma cadeia composta por três dígitos para que o mesmo case com o padrão, o\. define simplesmente o caractere "." e o símbolo $ informa o encerramento da mesma.

Sequencia	Negação	Descrição
\w	\W	Utilizado para casar letras, dígitos e _.
\d	\D	Utilizado para casar dígitos de 0 até 9.
\s	\S	Utilizado para casar espaços em branco
\b	\B	Utilizado para casar a separação de palavras, incluindo um começo e fim, ^ e $ respectivamente.
\A		Usado para casar o começo da cadeia de caracteres.
\Z		Usado para casar o fim da cadeia de caracteres.

Tabela 4. Sequências de Escape que podem ser utilizadas nas expressões

Caractere	Descrição
{n}	Utilizado para identificar o número especificado "n" de ocorrências na expressão.
{n,}	Utilizado para identificar uma o mais ocorrências de "n" na expressão.
{n,n2}	Aplicado para definir um número "n" para o número de ocorrências "n2" na expressão.
?	Zero ou nenhuma, ou {0,1}
*	Zero ou mais, o mesmo que utilizar {0,}
+	Uma ou mais, o mesmo que utilizar{1,}

Tabela 5. Caracteres utilizados para a Repetição

Outra expressão que pode ter bastante utilidade é a ^\d{2}-\d{4}-\d{4}$ para a validação de um telefone. Esta expressão pode ser traduzida como início, 2 dígitos, -, 4 dígitos, -, 4 dígitos e fim. Um fator que também deve ser levado em consideração ao trabalhar com expressões regulares é que existem diversas ferramentas que facilitam sua utilização, dentre elas editores de expressões e validadores para as mesmas (ver seção de links), gratuitas ou pagas.

Expressõesregularesna prática

Para dar início aos exemplos da utilização do recurso de Expressões Regulares no Delphi XE 2, basta criar um novo projeto "VCL FormsApplication". O próximo passo para utilização deste recurso é a declaração da Unit System.RegularExpressions na seção Uses do formulário. Dando continuidade ao projeto, a próxima etapa é inserir os componentes que serão utilizados para a aplicação deste recurso, sendo eles um TRadioGroup, para selecionar a expressão de validação, um TLabel, um TEdit, que conterá a String que será submetida a verificação, e para concluir um TSpeedButton que efetuará tal teste. Uma expressão regular nada mais é do que uma string que pode conter um ou mais símbolos que foram informados nas tabelas anteriores, sendo assim, para cria uma expressão regular basta utilizar uma constante ou variável do tipo String para receber a expressão e futuramente ser utilizada para a validação, ou se preferir, o desenvolvedor também pode criar um registro TRegEx que pode receber a expressão regular diretamente. O código da **Listagem 1** foi inserido no objeto SpeedButton1 e apresenta a primeira alternativa para utilização de expressões regulares.

Foi criada uma constante denominada email que recebe a expressão regular que será utilizada. Posteriormente o método IsMatch do TRegEx é chamado diretamente para a validação, lembrando que o mesmo retorna um valor booleano, sendo verdadeiro se a string casou com a expressão, ou falso, caso não exista o casamento. A **Listagem 2** demonstra uma outra forma de efetuar a mesma validação de email definida na listagem anterior, de uma maneira mais direta.

Listagem 1. Validar um e-mail

```
procedure TForm1.SpeedButton1Click(Sender: TObject);
const
  email = '^([0-9a-zA-Z]([-\.\w]*[0-9a-zA-Z])*@([0-9a-zA-Z][-\w]*[0-9a-zA-Z]\.)+
  [a-zA-Z]{2,9})$';
begin
if TRegEx.IsMatch(Edit1.Text, email) then
Application.MessageBox('A String informada casou com a RegEx!','RE', 64 + 0)
else
    Application.MessageBox('A String informada não casou com a RegEx!','RE', 64 + 0);
end;
```

Listagem 2. Expressão Regular criada em uma instância de TRegEx

```
procedure TForm1.SpeedButton2Click(Sender: TObject);
var
 RE: TRegEx;
begin
  RE := TRegex.Create('^([0-9a-zA-Z]([-\.\w]*[0-9a-zA-Z])*@([0-9a-zA-Z][-\w]*
  [0-9a-zA-Z]\.)+[a-zA-Z]{2,9})$');
if RE.IsMatch(Edit1.Text) then
Application.MessageBox('A String informada casou com a RegEx!','RE', 64 + 0)
else
    Application.MessageBox('A String informada não casou com a RegEx!','RE', 64 + 0);
end;
```

Foi declarada uma variável do tipo TRegEx e posteriormente a mesma é instanciada em conjunto com a definição da sua expressão regular através do método TRegex.Create(). Posteriormente é utilizado o método IsMatch() para o retorno booleano da validação.

Como pode ser observado em ambos os códigos, efetuar validações através de Expressões Regulares permitem um ganho de produtividade e de diversas possibilidades, uma vez que os desenvolvedores não necessitam criar diversos métodos para efetuar a verificação das informações, como por exemplo através do uso da função Pos(), que também pode ser utilizada várias vezes para verificar a presença de termos.

Contudo, para concluir esta etapaforam adicionados mais itens no objeto TRadioGroup, para permitir a utilização demais expressões para validação e o código presente na **Listagem 1** foi alterado para o código presente na **Listagem 3**. Isto com o intuito de demonstrar mais uma das grandes vantagens ao se utilizar RegEx, que permite que a validação de diversas informações sejam efetuadas utilizando apenas um TRegEx, uma vez que normalmente pode existir a necessidade de se criar uma função para validar cada campo. A **Listagem 3** apresenta a utilização de várias expressões regulares em conjunto com apenas um objeto TRegEx para utilizar a expressão de acordo com a seleção do objeto TRadioGroup.

Expressões regulares para pesquisa e substituição

Como já mencionado anteriormente, as Expressões Regulares são muito empregadas na pesquisa, filtragem e substituição de informações. Para dar início a esta etapa do exemplo em Delphi, o mesmo projeto criado anteriormente pode ser carregado e utilizado. Além do TRegEx, existem diversos objetos úteis que também podem ser utilizados em conjunto com o mesmo para auxiliar nas buscas e substituições, dentre eles o TMach, TMachEvaluator e TRegExOptions. O TMach é um registro que é responsável por conter um único resultado de acordo com a expressão regular. Ele possui seu próprio construtor, mas também pode ser construído através do método TRegEx.Match(), que retorna um valor deste tipo. Existe também o TMatchCollection, que é utilizado para armazenar diversas instâncias do TMatch.Neste caso o TMatchCollection pode ser obtido do método TRegEx.Matches().

Com relação ao TMatchEvaluator, ele permite a substituição das ocorrências de determinada expressão informada com relação a quantidade de vezes que a mesma foi encontrada no texto, podendo ser utilizado no método TRegEx.Replace().

No que concerne ao TRegExOption, o mesmo é um enumerador que deve ser utilizado para indicar as opções que deverão ser utilizadas pelo TRegEx. A **Tabela 6** demonstra as opções permitidas para um objeto TRegEx através do TRegExOption.

Conhecidos os objetos necessários para a busca e substituição de palavras, o próximo passo é dar início a utilização dos mesmos. Para tanto, foram incluídos mais dois TMemos, três TLabels, um TGroupBox, dois TCheckBoxs, um TEdit e um TSpeedButton para permitir a pesquisa de determinada palavra em um texto, de acordo com algumas opções definidas. Para efetuar tal pesquisa e também retornar um o número de ocorrências da expressão no texto, primeiramente foi criada uma variável global do tipo Integer denominada "Ocorrencias" para armazenar o número de ocorrências e posteriormente criado um procedimento recursivo para auxiliar na pesquisa.

Nota do DevMan

Um procedimento recursivo nada mais é que um procedimento que em determinado momento, chama a si mesmo. Este recurso é muito útil, pois permite que um código finito possa ser utilizado n vezes de acordo com a necessidade e o teste efetuado.

A **Listagem 4** demonstra o código do procedimento recursivo que será usado para identificar o número de ocorrências.

Como pode ser observado neste código, o procedimento recebe uma variável do tipo TMatch como parâmetro e adiciona em uma linha do MemoOcorrencias a mesma, incrementando também a variável "Ocorrencias". Posteriormente é utilizado o método NextMatch para ir para a próxima ocorrência e para finalizar, se a próxima ocorrência existir, o mesmo procedimento é chamado. A **Listagem 5** demonstra o código utilizado para efetuar a pesquisa através do recurso RegEx.

Listagem 3. Utilizando o TRegEx para efetuar multiplas validações

```
procedure TForm1.SpeedButton1Click(Sender: TObject);
const
  email = '^([0-9a-zA-Z]([-\.\w]*[0-9a-zA-Z])*@([0-9a-zA-Z][-\w]*[0-9a-zA-Z]\.)+
  [a-zA-Z]{2,9})$';
telefone = '^\d{2}\-\d{4}\-\d{4}$';
  databr = '^([0-9]|[0,1,2][0-9]|3[0,1])/([\d]|1[0,1,2])/\d{4}$';
  cpf = '^\d{3}\.\d{3}\.\d{3}\-?\d{2}$';
var
expressao: String;
begin
case RadioGroup1.ItemIndex of
0: expressao := email;
    1: expressao := telefone;
    2: expressao := databr;
else expressao := cpf;
end;
if TRegEx.IsMatch(Edit1.Text, expressao) then
Application.MessageBox('A String informada casou com a RegEx selecionada!','RE', 64 + 0)
else
   Application.MessageBox('A String informada não casou com a RegEx!','RE', 64 + 0);
end;
```

Listagem 4. Procedimento utilizado para auxiliar a pesquisa

```
procedure TForm1.AdicionarMatch(Match: TMatch);
begin
  MemoOcorrencias.Lines.Add(Match.Value);
  Inc(Ocorrencias);
  Match := Match.NextMatch;
if Match.Success then
    AdicionarMatch(Match);
end;
```

Após a declaração das variáveis, a variável "Ocorrencias" tem seu valor zerado para possibilitar a contagem. Posteriormente são verificados os objetos do tipo TCheckBox, e se os mesmos estiverem checado, são carregadas as opções para o TRegEx na

Opção	Descrição
roNone	Indica que nenhuma opção foi definida.
roIgnoreCase	Utilizado para indicar que o estado da letra deve ser ignorado, admitindo maiúsculas ou minúsculas, o mesmo que case-insensitive.
roMultiLine	Permite a utilização do modo Multilinha.
roExplicitCapture	Permite definir que as capturas válidas são determinados grupos informados.
roCompiled	Indica que a Expressão regular deve ser compilada para Assembly, o que torna a execução mais rápida, porém aumenta a duração da inicialização.
roSingleLine	Indica a utilização do modo de linha-única.
roIgnorePatternSpace	Utilizado para eliminar espaços em branco, permitindo também comentários utilizados com o símbolo #.

Tabela 6. Opções utilizadas no TRegEx através do TRegExOption

variável REOpcoes. Com as opções definidas, o próximo código é responsável pela criação do objeto RE do tipo TRegEx com a palavra que será pesquisada, presente no objeto EdPalavraChave, e com as opções habilitadas. No próximo trecho, o objeto Match recebe o resultado do método TRegEx.Match(), que contém a ocorrência retornada de acordo com o texto informado, neste caso o objeto MemoTexto. Na sequência se foi encontrado algum resultado, então a função AdicionarMatch() é chamada, senão simplesmente é exibida uma mensagem notificando que a palavra fornecida não foi encontrada no texto. Posteriormente se a variável Ocorrencias tiver seu valor maior que zero é exibida uma mensagem contendo o número de ocorrências. Para concluir, como pode ser observado, a maior parte do código é executada entre um try/except, evidentemente para que uma possível exceção (se houver) possa ser tratada ou simplesmente exibida.

Concluída a pesquisa, a próxima etapa é efetuar a substituição de palavras. Para efetuar a pesquisa em conjunto com a substituição de uma palavra utilizando o RegEx, um TMatchEvaluator, também presente na Unit System.RegularExpressions, pode ser utilizado. Neste processo de substituição foi criada umafunção retornando um valor em String para auxiliar na substituição dos termos. O código da **Listagem 6** apresenta a função auxiliar para a substituição que também incrementa a variável global Ocorrencias criada anteriormente.

Listagem 5. OnClick do TSpeedButton para efetuar a busca

```
procedure TForm1.SpeedButton4Click(Sender: TObject);
var
  RE: TRegEx;
  REOpcoes: TRegExOptions;
  Match: TMatch;
begin
  Ocorrencias := 0;
if CheIgnorarCase.Checked then
    Include(REOpcoes, TRegExOption.roIgnoreCase);
if CheMultiLine.Checked then
    Include(REOpcoes, TRegExOption.roMultiLine);
try
    RE := TRegEx.Create(EdPalavraChave.Text, REOpcoes);
    Match := RE.Match(MemoTexto.Text);
if Match.Success then
begin
      AdicionarMatch(Match);
end
   else
      Application.MessageBox('A palavra não foi encontrada!','RE', 64 + 0);
if Ocorrencias > 0 then
      ShowMessage('Foi(ram) encontrada(s) '+ IntToStr(Ocorrencias) +'
ocorrência(s)!');
except
on e: Exception do
      ShowMessage('Houve um erro: ' + e.Message);
end;
end;
```

Listagem 6. Função auxiliar para efetuar a substituição da palavra

```
function TForm1.Substituir(const Match: TMatch): string;
begin
  Inc(Ocorrencias);
  Result := EditSubstituirPor.Text;
end;
```

A função simplesmente recebe o objeto TMatch que contém o valor da expressão que será substituída, incrementa a variável Ocorrencias e por fim retorna o novo valor, ou em outras palavras, o termo que ocupará o lugar da palavra a ser substituída. Também é interessante ressaltar que esta função é repetida para cada termo que é encontrado no texto. A **Listagem 7** demonstra o código que foi inserido no evento OnClick do TSpeedButton adicionado anteriormente para efetuar a substituição das palavras de acordo com os valores preenchidos nos objetos TEdits.

Listagem 7. OnClick do TSpeedButton utilizado para Substituir uma palavra

```
procedure TForm1.SpeedButton3Click(Sender: TObject);
var
  RE: TRegEx;
  Avaliador: TMatchEvaluator;
begin
Ocorrencias := 0;
if (EditPalavraSubstituir.Text = '') or (EditSubstituirPor.Text = '') then
    Application.MessageBox('Informe a palavra que será substiuída e a que será
    atribuída!','RE', 64 + 0)
else
  begin
    RE.Create(EditPalavraSubstituir.Text);
Avaliador := Substituir;
    MemoTextoSubstituir.Text := RE.Replace(MemoTextoSubstituir.Text, Avaliador);
if Ocorrencias > 0 then
     ShowMessage('Foi(ram) substituída(s) '+ IntToStr(Ocorrencias) +' ocorrência(s)!')
    else
     Application.MessageBox('A palavra não foi encontrada!','RE', 64 + 0);
end;
end;
```

Foi efetuado um simples teste para verificar se os objetos TEdits foram preenchidos corretamente. Posteriormente é criadaa instância deTRegEx recebendo como parâmetro a palavra que será substituída, neste caso presente no EditPalavraSubstituir. Posteriormente, a variável Avaliador recebe a função que será utilizada para a substituição das ocorrências. Na sequência o texto do objeto MemoTextoSubstituir recebe o resultado da função Re.Replace(), relembrando que esta função é utilizada para efetuar a substituição de acordo com os parâmetros fornecidos, que neste caso são o texto que será substituído e o objeto Avaliador que efetuará a substituição). Para concluir, a variável Ocorrencias é testada para verificar se o valor de substituições é maior que zero bem como o número de vezes que ela ocorreu. Neste caso, é interessante ressaltar que o Avaliador pode ser opcional, ou seja, se o código utilizado fosse MemoTextoSubstituir.Text := RE.Replace(MemoTextoSubstituir.Text, EditSubstituirPor.Text), a substituição também ocorreria, porém a função foi criada para executar determinado trecho de código cada vez que a palavra é encontrada, possibilitando assim uma maior manipulação. A função StringReplace() também pode ser utilizada para a substituição de palavras ou caracteres, porém sua funcionalidade é bem limitada se comparada com a utilização de Expressões Regulares. A **Figura 1** apresenta a aparência final da aplicação, com todos os recursos apresentados neste artigo.

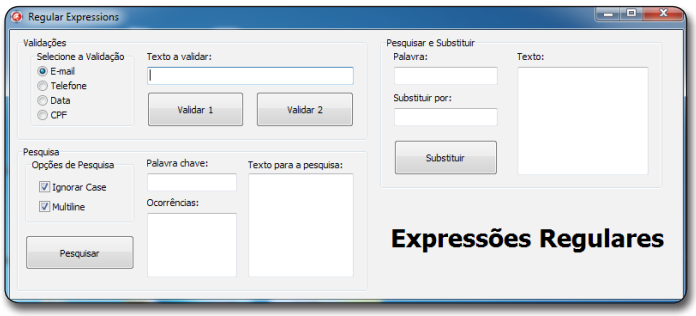


Figura 1. Visual final da aplicação de Expressões Regulares

Conclusão

O recurso de Expressões Regulares é um conceito que foi concebido há muito tempo, porém de muita utilidade e que pode ser amplamente aplicado na atualidade. Tal recurso permite um aumento da produtividade, uma vez que poupa esforços por parte dos desenvolvedores no que diz respeito à criação de funções compostas por diversos comandos e códigos para executar tarefas que com apenas alguns objetos e métodos, presentes na Unit RegularExpressions do Delphi XE 2, podem ser facilmente efetuadas. Como foi possível observar, através do uso de Expressões Regulares é possível efetuar a validação de campos, pesquisar palavras em um texto, substituir termos, entre outros, de uma maneira extremamente flexível, pois permite que o próprio desenvolvedor crie suas próprias expressões para validação de acordo com a necessidade do projeto em questão.

Entretanto, muitas vezes o recurso de Expressões Regulares não é amplamente explorado, sendo até deixado de lado, uma vez que na área de desenvolvimento de Software, assim como em outras áreas da Informática, sempre existem várias maneiras ou caminhos que levam ao mesmo objetivo, contudo vale à pena evidenciar que determinados "caminhos" podem oferecer uma maior possibilidade de recursos de uma maneira mais rápida e simplificada.

Giuliano Scombatti Pinto
giuliano@sygnux.com.br
É Analista Desenvolvedor da Sygnux Software (www.sygnux.com.br) localizada em Monte Alto/SP, atuando também com desenvolvimento Web e Android. Formado pela Fatec/TQ, começou a desenvolver com Delphi em 2003. Atualmente trabalha com Delphi, Java, PHP e Flash/Flex. Professor de Informática de Projetos Sociais da Prefeitura Municipal de Monte Alto/SP.

Informações sobre Expressões Regulares
http://www.regular-expressions.info/pcre.html

Validador de Expressões Regulares
http://www.regextester.com/

DevSpace do Autor
http://www.devmedia.com.br/space.asp?id=283274

Dê seu feedback sobre esta edição!

A Clubedelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre este artigo, através do link:

www.devmedia.com.br/clubedelphi/feedback

Seu **conhecimento vale muito** na DevMedia

Publique artigos e vídeos gratuitamente no site e participe do programa **PayContent.**

Sua participação pode ser premiada: quanto mais você postar, mais chances de ganhar.

Para saber mais acesse:
www.devmedia.com.br/paycontent

DevMedia

Aplicando Threads
Utilizando multithreads para re

Resumo DevMan

De que trata o artigo:

Este artigo apresenta uma breve introdução ao conceito de Threads e aplicações Multithread, um recurso que permite a execução de tarefas de maneira "paralela", dando a impressão de que as mesmas são efetuadas ao mesmo tempo em um ambiente com apenas um processador, demonstrando assim as vantagens que podem ser obtidas, bem como a facilidade na utilização deste recurso através do Delphi, apresentando também uma maneira de criar uma aplicação de download de arquivos em conjunto com Threads e a Internet.

Em que situação o tema é útil:

Aplicar o conceito de "simultaneidade" nas aplicações desenvolvidas para que as mesmas possam ser mais rápidas e eficientes, delegando às threads determinadas tarefas críticas que podem levar um maior tempo de execução, fazendo com que elas se encarreguem de tais tarefas, permitindo assim que o usuário possa fazer ações em paralelo na aplicação.

Threads e Multithread:

Atualmente a utilização de threads nas aplicações é um diferencial que traz diversas vantagens para o usuário final, uma vez que podem aumentar o desempenho da mesma. Elas permitem que as tarefas possam ser executadas em paralelo ao fluxo principal da aplicação, possibilitando assim que a mesma continue acessível ao usuário. O Delphi traz consigo diversas classes e recursos que podem ser manipulados para a utilização deste conceito nas aplicações de uma maneira simplificada e de alta produtividade, uma vez que em determinados tipos de softwares, a utilização de threads se torna uma obrigação para que o mesmo possa oferecer a funcionalidade esperada. Este artigo tem por objetivo efetuar uma introdução sobre threads e alguns de seus conceitos, demonstrando também como utilizar este recurso para criar uma aplicação que efetue o download de arquivos de maneira paralela à aplicação principal.

A computação é um ambiente que está em constate evolução. Há muitos anos atrás várias aplicações eram baseadas no conceito monotarefa e monousuário, onde apenas um software poderia ser executado por vez e por apenas um usuário, uma antiga realidade que nos dias de hoje, diante de tantos softwares executando "simultaneamente", parece inconcebível.

Thread é a unidade básica envolvida quando uma aplicação é executada em sistemas operacionais Unix, Windows, etc. Sendo assim, as threads que pertencem a determinado processo também são submetidas ao escalonamento, criando assim a impressão de que tudo é executado ao mesmo tempo com relação a um processador, devido ao rápido mecanismo de alternância envolvido nesta tarefa, salvo o fato de realmente executarem de maneira concorrente em múltiplos processadores. O escalonamento (scheduling) é uma atividade efetuada para permitir que diversos processos executem em paralelo com o objetivo de aumentar a produtividade, minimizando a sobrecarga e o tempo de resposta para os usuários, permitindo também que determinados processos possuam prioridades maiores que outros.

alização de downloads

Giuliano Scombatti Pinto

Um exemplo de técnica de escalonamento é a implementação do recurso time slice (fatia de tempo), onde cada thread é alocada e executada pelo processador em um determinado período de tempo gerando tal impressão de "simultaneidade". Contudo existem os ambientes Monothread e Multithread.

Em um ambiente Monothread, uma aplicação é composta por códigos que são executados sequencialmente, de acordo com repetições, desvios, funções e procedimentos, onde a utilização de concorrência é efetuada através da implementação de processos independentes e subprocessos, caso exista essa necessidade, causando um maior consumo de recursos do sistema, uma vez que cada processo detém seu próprio espaço de endereçamento e a comunicação entre os diversos processos envolvidos normalmente é mais lenta que no outro modo.

Em um ambiente Multithread, o conceito de thread é amplamente explorado e aplicado, onde técnicas permitem que as mesmas sejam executadas de maneira simultânea. Através da utilização deste recurso, os softwares podem efetuar diversas tarefas de maneira paralela, possibilitando assim que as threads exerçam suas funções de maneira independente umas das outras. Vale à pena evidenciar que os sistemas operacionais evoluíram para possibilitar a execução de diversos processos e threads, implementando mecanismos para que os softwares pudessem fazer uso de tal recurso de concorrência, como é o exemplo do escalonamento citado anteriormente. A utilização de várias threads é mais rápida que a utilização de vários subprocessos, pois as threads de um determinado processo compartilham o mesmo intervalo de endereço (address space), permitindo assim a troca de diversas informações entre ambas, não havendo a necessidade da utilização de outros mecanismos para a intercomunicação, como é o caso dos processos.

Apesar de sua popularidade no universo computacional, a utilização de threads muitas vezes não é adotada nas aplicações, seja para não fugir do modelo convencional de desenvolvimento ou por receio, contudo a utilização deste recurso é simples, principalmente no Delphi. Entretanto, assim como outras funcionalidades, alguns cuidados devem ser tomados ao fazer a implementação deste recurso, levando também em consideração que as threads devem manter a comunicação, o compartilhamento e a sincronização de dados de forma coerente. Sendo assim, uma boa prática na utilização de threads é trabalhar com variáveis e objetos privados, utilizando funções e procedimentos para o acesso aos mesmos, tomando cuidado com a comunicação e a sincronização das mesmas. Também vale a pena evidenciar que não há um limite lógico para a utilização de threads simultaneamente, entretanto os limites de hardware e Internet devem ser levados em consideração para não causar sobrecarga desnecessária ao sistema.

Níveis, organização, integração

As threads podem ser organizadas por nível, Temos o tipo ULT (User-Level Thread), KLT (Kernel-Level Thread) ou Modo Híbrido. Nas Threads do tipo ULT, em nível de usuário, o processo de escalonamento é feito pelo próprio desenvolvedor, tirando a obrigação do sistema operacional em gerenciá-las e fazendo com que as mesmas sejam mais rápidas e eficientes, porém mais difíceis de serem implementadas. Já no KLT, em nível de Kernel (Núcleo do Sistema), o processo de escalonamento é feito pelo sistema operacional, o que torna este nível mais lento que o ULT, porém é mais fácil de ser implementado, uma vez que o sistema se encarrega de oferecer os mecanismos necessários para o gerenciamento das threads. No que diz respeito ao Modo Híbrido, como seu próprio nome sugere, a intenção é a combinação das vantagens de ambos os níveis, ULT e KLT. Este modo apresenta uma flexibilidade superior, porém também herda algumas das desvantagens presentes nas outras implementações.

Com relação à organização de sistemas multithread, normalmente existe uma thread que é a principal e responsável pela implementação dos mecanismos, efetuando o gerenciamento das outras threads envolvidas. Na década de 90, em meio a diversas dificuldades com relação à utilização das threads, foi estabelecido um padrão POSIX (Portable Operating System Interface) que ficou conhecido como Pthreads, fazendo com que os softwares que implementassem multithread fossem padronizados, principalmente nos ambientes operacionais Unix Like (Unix e derivados). Tal modelo também permite a implementação dos modos ULT, KLT e Híbrido.

Sobre a integração entre aplicações e conexões utilizando a Internet, as threads podem oferecer uma base fundamental para o funcionamento das mesmas, uma vez que em determinadas situações, as solicitações efetuadas através da Internet e seus retornos podem ser demorados, o que não convém fazer com que a Thread principal aguarde a conclusão da atividade, fazendo também com o usuário tenha que esperar a finalização desta etapa para somente então continuar utilizando o programa, não implementando assim a simultaneidade.

Sendo assim, a utilização de threads traz muitos benefícios às aplicações, pois permite dividir as tarefas, aumentar o tempo de resposta e também o monitoramento dos eventos que estão sendo executados.

Evento/Propriedade	Descrição
Create	Utilizado para a criação da Thread, onde o estado da mesma deve ser informado no parâmetro denominado CreateSuspended. Se TThread.Create(True), então a Thread não é iniciada automaticamente, senão ela é executada a partir deste momento.
Priority	Como seu próprio nome sugere, permite a determinação do nível de prioridade de execução da Thread em questão com relação as outras Threads que fazem parte do processo atual. Sendo assim, ao aumentar o nível de prioridade, o tempo de processamento dedicado a Thread também é aumentado.
Resume	Faz com que a Thread inicie, caso tenha sido criada em estado de suspensão, efetuando assim o código presente no procedimento Execute.
Suspend	Permite que a Thread seja pausada para que sua execução possa ocorrer posteriormente.
Syncronize	Responsável pela sincronização da Thread com a Thread principal da aplicação, efetuando uma chamada a um procedimento externo. Uma boa prática é utilizá-lo todas às vezes em que um evento que pertence a um objeto que não faz parte da Thread for chamado.
Execute	O código que será executado pela Thread. Uma boa prática é efetuar a verificação do valor da propriedade Terminated, pois se o mesmo for True indica que a Thread foi finalizada.
OnTerminate	É efetuado quando a execução da Thread é encerrada.
FreeOnTerminate	Quando definido como True, permite a destruição/liberação da instância de TThread de uma forma automática, dispensando assim o uso do FreeAndNil(), que é utilizado para liberar objetos.
Terminate	Faz com que a execução da Thread seja encerrada.

Tabela 1. Métodos e Eventos da classe TThread do Delphi

Delphi e threads

No Delphi, quando o projeto é executado é iniciada uma thread que contém o processo principal da aplicação. Sendo assim, quando são efetuadas as consultas ou procedimentos extensos e demorados, é possível observar que não há a possibilidade de fazer outra tarefa neste mesmo programa enquanto a atual solicitação não é concluída. Neste momento é notável a vantagem ao adotar a utilização de Multithread nos projetos, pois assim seria possível efetuar uma determinada tarefa enquanto outra (ou outras) são efetuadas. O componente TTimer do Delphi é um simples exemplo da utilização de Thread, pois permite que uma tarefa seja executada em paralelo com o fluxo principal do programa. O Delphi conta com uma classe denominada TThread que permite a implementação de uma Thread. Esta classe pode ser herdada para que os desenvolvedores utilizem este recurso de acordo com suas necessidades. Para tanto, também convém estender/sobrescrever (override) o procedimento Execute, um método abstrato que precisa ser implementado, presente na classe TThread, fazendo com que o mesmo possa ser utilizado para qualquer fim. As threads no ambiente do Delphi contam com diversos eventos que são úteis e importantes na manipulação das mesmas. A **Tabela 1** apresenta alguns eventos e propriedades.

Inciando o projeto

Como é possível notar com as informações descritas anteriormente, as threads também podem ser utilizadas com o propósito de dividir tarefas maiores em menores e assim fazer com um determinado objetivo que demoraria muito tempo para ser alcançado, possa ser atingido de uma maneira mais rápida. Desta maneira, o conceito de sincronização entre Threads se torna essencial. Para dar início ao exemplo de uma aplicação Multithread em conjunto com conexões, será criado um gerenciador de donwloads onde o usuário pode efetuar vários downloads, utilizando Threads, sem que o programa principal fique indisponível durante este processo. Sendo assim, basta criar um novo projeto VCL Forms Application. Para a execução deste exemplo foi inserido um objeto TSaveDialog, um TProgressBar, um TStringGrid e um TSpeedButton. Também será utilizado neste projeto um objeto do tipo TCriticalSection, presente na Unit SyncObjs. Um objeto do tipo TCriticalSection, assim como seu nome sugere, deve ser utilizado para a implementação de uma seção crítica, o que significa que em um ambiente Multithread, este recurso é utilizado para garantir que uma thread não execute determinadas áreas dos códigos antes que uma operação efetuada por outra thread seja concluída, protegendo e bloqueando assim determinadas seções. Contudo, o Delphi também conta com o objeto TThreadList, que também permite implementar um controle para evitar problemas em aplicações Multithread. Dentre os métodos deste objeto que podem ser utilizados para estes fins, o primeiro é denominado Lock, que faz com que o acesso seja bloqueado, e o segundo é denominado Unlock, que faz a liberação. Dando continuidade ao projeto, foi criada uma variável Global privada no formulário principal do tipo TCriticalSection que foi denominada "CS", que será utilizada para a implementação do mecanismo descrito anteriormente, bem como um simples procedimento público para que a variável CS possa ser utilizada por outras Units. Vale à pena evidenciar que neste projeto, o conceito de encapsulamento também foi adotado, uma vez que a Thread que será criada futuramente em uma Unit a parte, efetuará diversos acessos aos objetos e variáveis do formulário principal para as atualizações das informações. O encapsulamento é uma técnica muito importante presente na Orientação a Objetos, utilizada para "esconder" variáveis, objetos ou determinados métodos privados da classe em questão, onde normalmente são criados métodos públicos para possibilitar que tais objetos privados possam ser acessados (get) ou alterados (set) por outras classes, sem que seja necessário conhecer o funcionamento interno dos mesmos, em outras palavras, é a abstração das variáveis, objetos e métodos por parte da classe. Também foi criada outra variável global privada do tipo Integer para armazenar o número de downloads que estão em andamento e foi

denominada de "NumeroDownload". A **Listagem 1** demonstra um simples procedimento que foi criado no formulário principal com o objetivo de iniciar e liberar o uso da variável CS através da utilização de um procedimento público.

Listagem 1. Procedimento para a utilização da variável CS por outras Units

```
procedure TFoPrincipal.setCriticalSection(Ativar: Boolean);
begin
  if Ativar then
    CS.Acquire
  else
    CS.Release;
end;
```

Como é possível observar neste código, este procedimento foi utilizado para que futuramente as threads ou outras Units não façam o acesso direto à variável, mas efetuem uma chamada a este procedimento público. Contudo, esta variável poderia ser declarada como pública caso o objetivo fosse acessá-la externamente de maneira direta. Sendo assim, se o parâmetro que foi denominado Ativar for definido como True, o método Acquire é utilizado. Este método é utilizado para efetuar o vínculo entre o segmento de chamada e a seção crítica, fazendo com que os outros segmentos sejam bloqueados. Já o método Release efetua a liberação, permitindo assim que os outros seguimentos possam utilizar a seção crítica. Também foram criados os códigos para a inicialização da variável e definições do objeto TStringGrid assim que o formulário for criado, bem como a destruição da variável CS em conjunto com o formulário. A **Listagem 2** apresenta a inicialização e destruição da variável CS e a definição de algumas propriedades o objeto TStringGrid.

Listagem 2. Criação e destruição da variável CS em conjunto com o formulário

```
procedure TFoPrincipal.FormCreate(Sender: TObject);
begin
  CS := TCriticalSection.Create;
  NumeroDownload := 0;
  with StringGrid1 do
  begin
    ColWidths[0] := 10;
    ColWidths[1] := 200;
    ColWidths[2] := 100;
    ColWidths[3] := 100;
  end;
end;

procedure TFoPrincipal.FormDestroy(Sender: TObject);
begin
  CS.Free;
end;
```

Como pode ser observado, no evento OnCreate do formulário o objeto TStringGrid tem suas colunas ajustadas, bem como o objeto TCriticalSection é instanciado, sendo liberado no evento OnDestroy.

A próxima etapa neste projeto é a criação de uma nova Unit apenas para separar os códigos que pertencem ao objeto TThread, que será herdado e estendido, dos códigos e objetos do formulário principal. Sendo assim, foram incluídas na seção Uses da mesma as seguintes Units: Classes, System.SysUtils, IdHTTP, IdComponent, Vcl.Dialogs, Vcl.Controls e UPrincipal (A Unit que contém o formulário utilizado neste projeto). Então foi criada a classe que herda a TThread. A **Listagem 3** demonstra o código utilizado para criar a classe TDownload, que por sua vez herda a classe TThread do Delphi.

A classe TDownload herda de TThread. Posteriormente, o método Execute, um método abstrato que deve conter o código que será executado pela Thread, é estendido através do override. O override permite que os procedimentos da classe pai possam ser estendidos ou sobrescritos na classe filha, permitindo que os mesmos possam ser adaptados de acordo com a necessidade do projeto. Em seguida, é criado um procedimento que foi denominado Download(), cujo objetivo será iniciar o download do arquivo através da URL informada, bem como o local onde o arquivo será gravado. Logo em seguida, foram criados mais três procedimentos que serão utilizados futuramente para atualizar o status do download em andamento. Também foram criadas algumas variáveis privadas com o intuito de armazenar as informações que serão definidas através do procedimento público que vem na sequência, SetDownload().

Listagem 3. Código utilizado para a criação da classe TDownload

```
type
  { Thread }
  TDownlaod = class(TThread)
    protected
      procedure Execute; override;
      procedure Download(Url, SalvarEm: string);
      class procedure Work(ASender: TObject; AWorkMode: TWorkMode;
       AWorkCount: Int64);
      class procedure WorkBegin(ASender: TObject; AWorkMode: TWorkMode;
       AWorkCountMax: Int64);
      class procedure WorkEnd(ASender: TObject; AWorkMode: TWorkMode);
    private
      FUrl: String;
      FArquivo: String;
      FNumero: Integer;
    public
      procedure SetDownload(Numero: Integer; Url, SalvarEm: string);
  end;
```

O primeiro procedimento que implementado é o SetDownload(), que é utilizado para a configuração das variáveis globais privadas desta Unit. A **Listagem 4** demonstra o código do procedimento SetDownload() da thread TDownload.

As variáveis globais privadas são definidas para que futuramente sejam acessadas. A seguir temos o procedimento Download(), que será responsável por efetuar o download do arquivo informado no destino que foi escolhido. Para este procedimento serão utilizados mais dois componentes, o TIdHTTP e o TFileStream. O objeto TIdHTTP, pertencente ao grupo Indy Clients, é uma classe que descende do TIdCustomHTTP e tem por objetivo implementar um cliente para o protocolo HTTP . Este componente também conta com diversas propriedades e eventos, entre eles o WorkBegin, Work e WorkEnd, que respectivamente são utilizados para a execução de determinado trecho de código no início da tarefa, durante e no final da mesma, permitindo assim ao desenvolvedor detalhar o que deve ser feito a cada etapa. Outro procedimento muito importante deste mesmo objeto é o Get(), que permite o acesso a uma determinada URL, tendo seu retorno gravado em uma variável. Já o objeto TFileStream permite que a aplicação efetue operações de acesso ao disco para leitura e/ou gravação de arquivos, possuindo também diversos métodos úteis. A **Listagem 5** apresenta a implementação do procedimento Download() com a utilização dos objetos descritos anteriormente.

Nota do DevMan

A URL (Uniform Resourse Locator) é o termo utilizado para definir o endereço de um determinado recurso ou site presente em uma rede. Através da URL também é possível saber qual o protocolo que será utilizado, por exemplo, a URL "http://www.endereco.com.br", além de definir o endereço do recurso, indica que o protocolo utilizado é o HTTP.

Nota do DevMan

O HTTP (Hypertext Transfer Protocol), protocolo de transferência de texto dinâmico, é o protocolo padrão para a transferência de dados em uma rede ou na Internet, sendo o mesmo utilizado para o acesso a um website.

Listagem 4. Código do procedimento SetDownload() da Thread TDownload

```
procedure TDownlaod.SetDownload(Numero: Integer; Url, SalvarEm: string);
begin
  Self.FNumero := Numero;
  Self.FUrl := Url;
  Self.FArquivo := SalvarEm;
end;
```

Listagem 5. Código do procedimento Download()

```
procedure TDownlaod.Download(Url, SalvarEm: string);
var
  ArquivoDownload: TFileStream;
  IndyHTTP: TIdHTTP;
begin
  IndyHTTP := TIdHTTP.Create(FoPrincipal);
  try
    ArquivoDownload := TFileStream.Create(SalvarEm, fmCreate);
    try
      with IndyHTTP do
      begin
        OnWork := Work;
        OnWorkBegin := WorkBegin;
        OnWorkEnd := WorkEnd;
      end;
      IndyHTTP.Get(Url, ArquivoDownload);
    finally
      FreeAndNil(ArquivoDownload);
      FreeAndNil(IndyHTTP);
    end;
  except
    on e: Exception do
      ShowMessage('Houve um erro: '+ e.Message);
  end;
end;
```

O objeto IndyHTTP do tipo TIdHTTP é criado. Na sequência, também é criado o TFileStream (ArquivoDownload) indicando o local onde o mesmo será gravado de acordo com o parâmetro informado. No trecho seguinte, são definidos os eventos para a atualização do objeto ProgressBar1 presente no formulário principal. Posteriormente é utilizado o procedimento Get() do objeto IndyHTTP, fazendo com que o download inicie e seu conteúdo seja enviado para o objeto ArquivoDownload. Para finalizar, ambos os objetos são liberados da memória e também é incluído o código responsável pela manipulação das exceções através do try/except.

Como observado, foram definidos alguns eventos para que o objeto ProgressBar1 fosse atualizado de acordo com a evolução do download. Porém antes da demonstração do código destes eventos, deve ser criado o procedimento setBarraProgresso() no formulário principal para efetuar a atualização do estado da barra de progresso. A **Listagem 6** demonstra o procedimento criado para esta atualização.

São alteradas as propriedades Position e Max da barra de progresso (ProBarDownload1) de acordo com as variáveis que serão informadas pela Thread. Concluída esta etapa, agora serão implementados tais processos mencionados na Thread, lembrando que os mesmos são utilizados pelo objeto IndyHTTP. A **Listagem 7** demonstra os três procedimentos que foram criados neste exemplo para controlar a evolução da barra de progresso do formulário principal.

Listagem 6. Procedimento criado para o ajuste do TProgressBar

```
procedure TFoPrincipal.setBarraProgresso(Posicao, Maximo: Integer);
begin
  with ProBarDownload1 do
  begin
    if Posicao > 0 then
      Position := Posicao
    else
      Posicao := Max;
    if Maximo > 0 then
      Max := Maximo
  end;
end;
```

Listagem 7. Atualização do TProgressBar através da Thread

```
class procedure TDownlaod.WorkBegin(ASender: TObject; AWorkMode: TWorkMode;
  AWorkCountMax: Int64);
begin
  FoPrincipal.setBarraProgresso(0, AWorkCountMax);
end;
class procedure TDownlaod.Work(ASender: TObject; AWorkMode: TWorkMode;
  AWorkCount: Int64);
begin
  FoPrincipal.setBarraProgresso(AWorkCount, 0);
end;
class procedure TDownlaod.WorkEnd(ASender: TObject; AWorkMode: TWorkMo-
de);
begin
  FoPrincipal.setBarraProgresso(0, 0);
end;
```

Listagem 8. Procedimento Execute da Thread TDownload

```
procedure TDownlaod.Execute;
begin
  inherited;
  Priority := tpNormal;
  try
    FoPrincipal.setCriticalSection(True);
    if (FUrl <> '') and (FArquivo <> '') then
      Download(FUrl, FArquivo);
  finally
    FoPrincipal.setCriticalSection(False);
    FoPrincipal.setFinalizado(FNumero);
    ShowMessage('Download Completo!');
  end;
end;
```

O primeiro evento é executado no momento do início do download, o segundo é executado durante e o terceiro é executado no final do mesmo, como o objetivo de manter o progresso do download atualizado. Também é possível notar que o procedimento setBarraProgresso() é chamado em ambos os eventos com seus parâmetros definidos de acordo com a necessidade de ambos.

Finalizados os novos procedimentos criados para a Thread TDownload, a próxima etapa é a extensão do procedimento Execute da mesma, lembrando que este procedimento é o que será executado no momento em que a Thread for inicializada no Create ou no Resume. A **Listagem 8** demonstra o código que foi utilizado no procedimento Execute da Thread TDownload.

O nível de prioridade da Thread é definido como normal (tpNormal). Posteriormente, o procedimento setCriticalSection() criado anteriormente é utilizado para ativar o objeto TCriticalSection do formulário principal, sendo também efetuada uma simples verificação das variáveis e na sequência o procediemnto Download() é chamado. Para concluir, o procedimento setCritical Section() é chamado novamente para liberar o objeto TCriticalSecion. O procedimento setFinalizado() também é chamado e uma simples mensagem é exibida informando que o download do arquivo foi concluído.

Com estes passos, os códigos da Unit que contém a Thread TDownload são concluídos. Agora é preciso criar uma instância da Thread TDwonload e imediatamente iniciar o download mediante as informações que são solicitadas. A **Listagem 9** demonstra o código que foi inserido no TSpeedButton para a criação da thread e consequentemente, o início do download.

A função InputQuery() é utilizada para apresentar a caixa de diálogo de entrada de informações para permitir que o usuário forneça o Link para o download do arquivo desejado. Posteriormente é feito um teste para verificar se a variável UrlArquivo foi preenchida para dar sequência. Então a variável Arquivo recebe o nome contido na variável UrlArquivo e na sequência é utilizada pelo objeto SaveDialog1 como sugestão para o nome do arquivo (FileName). Sendo assim, o SaveDialog1 é executado para que o usuário possa selecionar o local onde o arquivo será gravado. Se arquivo foi definido no SaveDialog1, então a Thread Dw, do tipo TDownload, é criada de maneira suspensa.

Posteriormente é definido o valor de FreeOnTerminate como verdadeiro para que a Thread seja liberada da memória no final da execução. O procedimento SetDownload() da Thread Dw é utilizado para configurar as variáveis da mesma que serão utilizadas para indicar as informações do download. Para concluir, é chamado o procedimento Resume, fazendo com a Thread seja inicializada e posteriormente dê início ao download de maneira paralela.

Para concluir este exemplo foi criado o procedimento setFinalizado() no formulário principal que será chamado pela thread assim que o download é concluído, informando o número que foi atribuído a ela e fazendo com que o StringGrid1 seja atualizado.

Listagem 9. Código do evento OnClick do TSpeedButton para a criação da Thread e início do download

```
procedure TFoPrincipal.BoEfetuarDownloadClick(Sender: TObject);
  UrlArquivo, Arquivo: String;
  Dw: TDownlaod;
begin
  InputQuery('Download de Arquivo','Informe a URL:',UrlArquivo);
  if UrlArquivo <> '' then
  begin
    Arquivo := ExtractFileName(StringReplace(UrlArquivo,'/','\',[rfReplaceAll]));
    SaveDialog1.FileName := Arquivo;
    if SaveDialog1.Execute then
      if SaveDialog1.FileName <> '' then
      begin
        Dw := TDownlaod.Create(True);
        try
          with Dw do
          begin
            FreeOnTerminate := True;
            SetDownload(NumeroDownload, UrlArquivo, SaveDialog1.FileName);
            Resume;
          end;
          with StringGrid1 do
          begin
            RowCount := NumeroDownload +1;
            Cells[1, NumeroDownload] := Arquivo;
            Cells[2, NumeroDownload] := 'Inicializado!';
          end;
        finally
          Inc(NumeroDownload);
        end;
      end;
  end
  else
    ShowMessage('Informe a URL!');
end;
```

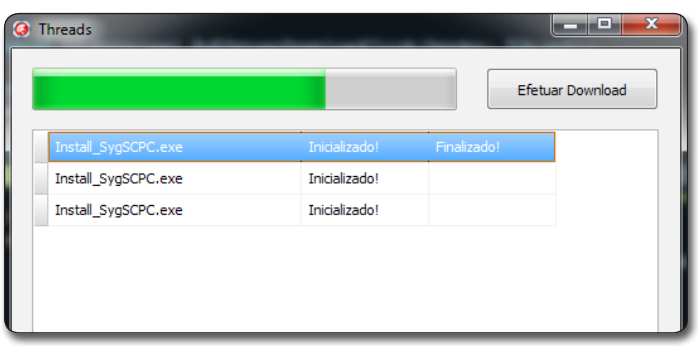


Figura 1. Visual final da aplicação de gerenciamento de download

A utilização de Threads é um recurso muito útil e necessário em aplicações que efetuam conexões externas, como neste caso, ou em softwares mais complexos que necessitem que diversas tarefas sejam executadas ao mesmo tempo. A **Figura 1** apresenta o visual final da aplicação de Download com a utilização de Threads.

Conclusão

O conceito de Threads foi concebido há mais de 20 anos e é muito aplicado atualmente nas aplicações. Entretanto muitas vezes este recurso acaba sendo esquecido de ser utilizado nos projetos. Como visto, através da utilização do recurso de Multitrhead uma aplicação consegue efetuar diversas tarefas concorrentes de uma maneira simplificada. As Threads permitem um maior desempenho com relação à utilização de diversos processos. O Delphi implementa diversas funcionalidades e classes para a utilização de Threads nas aplicações. Entretanto, vale à pena lembrar que na maioria das vezes, um determinado tipo de aplicação não necessita que Threads sejam iniciadas a todo o momento, mas somente em determinados momentos específicos da mesma. O uso de Threads se torna um diferencial, como é o caso da utilização das Threads para efetuar conexões e obter conteúdo da Internet, bem como em uma aplicação que implemente a função de copiar arquivos, lembrando que a utilização de várias threads permite que o usuário efetue outras tarefas enquanto a thread é executada em "paralelo".

Para concluir, o uso de Threads traz grandes benefícios tanto para os desenvolvedores, uma vez que os mesmos podem criar aplicações mais sofisticadas, quanto para os usuários, que desfrutam da possibilidade de efetuar diversas tarefas ao mesmo tempo no software.

Giuliano Scombatti Pinto
giuliano@sygnux.com.br
É Analista Desenvolvedor da Sygnux Software (www.sygnux.com.br) localizada em Monte Alto/SP, atuando também com desenvolvimento Web e Android. Formado pela Fatec/TQ, começou a desenvolver com Delphi em 2003. Atualmente trabalha com Delphi, Java, PHP e Flash/Flex. Professor de Informática de Projetos Sociais da Prefeitura Municipal de Monte Alto/SP.

Informações a respeito das Threads no ambiente Windows
http://msdn.microsoft.com/en-us/library/system.threading.thread.aspx

Informações sobre Threads (classe TThread) no Delphi
http://docwiki.embarcadero.com/VCL/en/Classes.TThread

DevSpace do Autor
http://www.devmedia.com.br/space.asp?id=283274

Dê seu feedback sobre esta edição!

A Clubedelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre este artigo, através do link:

www.devmedia.com.br/clubedelphi/feedback

FireMonkey e NFe 2.0

Uma solução passo a passo – Parte 2

Giuliano Scombatti Pinto

Resumo DevMan

De que trata o artigo:

Esta segunda parte do mini curso de NF-e com Delphi XE 2 utilizando a Firemonkey aborda o uso dos certificados digitais, assinatura e validação utilizando o componente CAPICOM, responsável por prover recursos de criptografia às aplicações. Para concluir, é apresentado um exemplo de consumo do WebService de Status de Serviço da Secretaria da Fazenda.

Em que situação o tema é útil:

Possibilitar ao desenvolvedor conhecimentos sobre como obter acesso ao certificado digital da entidade através do componente CAPICOM e do Firemonkey e assim, assinar as notas fiscais, vale destacar que tais recursos podem ser utilizados para assinar outros documentos também através da aplicação. Também é apresentada a biblioteca MSXML para auxiliar na manipulação do documento XML através de DOM, facilitando no momento da validação do mesmo.

FireMonkey e NFe 2.0:

As Notas Fiscais Eletrônicas atualmente são uma obrigatoriedade nas aplicações comerciais. Nesta segunda parte do artigo é abordada a utilização de certificados na aplicação FireMonkey através da biblioteca CAPICOM, uma tecnologia desenvolvida pela Microsoft para prover recursos relacionados à segurança nas aplicações, tendo como objetivo neste projeto a assinatura das notas fiscais eletrônicas. Também é apresentado o componente MSXML que provê recursos para o mapeamento de documentos XML de uma maneira simples, facilitando o momento da validação da NF-e. Para concluir, é abordada uma maneira de consumir o WebService de status da Receita para verificar se o serviço está em operação.

Na primeira parte desta série de artigos foram abordadas as estruturas básicas necessárias para a elaboração da NF-e, como a geração do documento XML correspondente à mesma, adotando o recurso XML Data Binding em conjunto com os schemas fornecidos pela própria Secretaria da Fazenda, bem como foram criadas as funções necessárias para a formatação de caracteres, geração do número aleatório, cálculo do dígito verificador e elaboração da chave de acesso. Vale à pena lembrar que o recurso XML Data Binding é muito importante para a manipulação de documentos XML porque permite através de seu assistente visual gerar classes tipadas a partir de documentos XML e, consequentemente, da NF-e através da importação dos esquemas da SEFAZ para a geração das classes.

Como a NF-e é um documento eletrônico e trafega através da Internet, foram adotadas algumas exigências e padrões para que a autenticidade da mesma fosse confirmada, como é o caso da assinatura digital que é obtida através de um certificado digital exclusivo da entidade. A Nota também deve ser validada no que diz respeito a sua estrutura, comparando se a mesma está de acordo com o XMLSchema correspondente, evitando assim erros e o envio de documentos divergentes ou incompletos. Para lembrar, os schemas (arquivos XSD – XML Schema Definition) são documentos baseados em XML que possuem regras de validação a serem aplicadas em arquivos XML

Nota do DevMan

A criptografia é a união de diversas técnicas e conceitos para codificar uma determinada informação, tornando-a confidencial, de uma maneira que somente a entidade emissora e a receptora possam "entender" a mesma, não permitindo que outros tenham acesso direto a informação. As técnicas de criptografia computacional geralmente são baseadas na utilização de chaves criptográficas.

Nota do DevMan

O Certificate Store no Windows é o local onde o mesmo armazena os certificados digitais instalados. Existem três tipos certificate store: User Account, Service Account e My Computer. User Account armazena certificados registrados para o usuário do Windows, Service Account refere-se a certificados registrados para serviços específicos e My Computer, para certificados exclusivos da máquina, não compartilhados. Cada um desses contém um número de pastas que representam os tipos de certificados existentes. Para gerenciar esse certificate store o Windows disponibiliza uma ferramenta que pode ser acessada pela seguinte linha de comando: mm\windows\system32\certmgr.msc. Este mostra de forma estrutura quais os certificados instalados no computador.

Concluído o trâmite de tais processos, o próximo passo é iniciar a comunicação com os WebServices da SEFAZ. Ratificando que existem WebServices específicos para cada serviço oferecido. O primeiro WebService que pode ser consumido pela aplicação NF-e é o WebService de status, para obter o estado atual do serviço, ou em outras palavras, para verificar se determinada nota já pode ser enviada ou consultada.

Certificado digital e assinatura

A Internet possibilita as mais diversas trocas de informações entre as entidades de uma maneira muito rápida, independentemente da distância física entre ambas. Sendo assim, é conveniente utilizar a Internet para efetuar transações eletrônicas. Contudo, ela é um meio onde podem existir diversas instabilidades ou até mesmo fraudes. Para garantir a autenticidade e integridade das transações eletrônicas, tornando-as mais seguras, existe a certificação digital. A certificação digital é composta por um certificado digital e consequentemente uma assinatura. No caso da emissão de uma Nota Fiscal Eletrônica válida juridicamente, também existe a necessidade da utilização destes recursos. A assinatura digital é simplesmente um conjunto de dados que é anexado a um documento (neste caso o XML da NF-e) para a identificação do emitente do mesmo, comprovando assim sua validade. Tal documento é criptografado através da chave privada do emitente, gerando assim uma assinatura. Para efetuar uma assinatura digital da NF-e, a empresa que emitirá a nota deve obter um certificado digital com uma autoridade certificadora credenciada à ICP-Brasil (Infraestrutura de Chaves Públicas Brasileira). O certificado pode ser do tipo A1 ou A3 e deve conter informações como o CNPJ do estabelecimento emitente. A **Tabela 1** descreve detalhadamente os tipos de certificados utilizados para a assinatura da NF-e.

A utilização do certificado digital é aplicada em dois momentos no processo de emissão da Nota Fiscal Eletrônica. Em um primeiro momento para a assinatura do documento XML, e posteriormente para efetuar a transmissão da mesma através dos WebServices. A assinatura da nota é efetuada através da tag "infNFe", sendo identificada pelo atributo Id. Os padrões adotados para a assinatura digital da nota fiscal eletrônica são: XML Digital Signature utilizando o formato Enveloped, certificado digital X.509 versão 3 do tipo A1 ou A3 contendo chave privada de 1024 bits, utilizando criptografia assimétrica RSA, message digest SHA-1, transformações C14N e Enveloped.

Assinando documentos com CAPICOM

Como foi mencionado, para assinar um documento existe a necessidade da obtenção de um certificado digital. Depois de obtido, para acessar tal certificado, pode-se acessar o Certificate Store do Windows.

Neste momento, a biblioteca CAPICOM (Cryptographic Application Programing Interface) pode ser utilizada através do Delphi para efetuar tal processo. A CAPICOM é uma biblioteca desenvolvida pela Microsoft para a plataforma Windows, na qual possui suporte a uma interface COM (Component Object Model) que permite a utilização de funções da CryptoAPI (Cryptographic Application Programming Interface).

Por ser um componente ActiveX, pode ser utilizada por qualquer ambiente que ofereça suporte a tal tecnologia, como é o caso do Delphi. Ela fornece aos desenvolvedores a possibilidade de autenticar e utilizar assinaturas digitais em documentos, criptografar e descriptografar dados, adicionar e remover certificados através da CryptoAPI, como mencionado anteriormente, permitindo assim a utilização de recursos de segurança nas aplicações. Ela pode ser utilizada para a assinatura de dados em formato texto ou binário. Possui suporte a certificados onde a chave primária pode estar armazenada em um Smart Card, Token USB ou Software. O certificado pode ser selecionado automaticamente pela aplicação ou pelo usuário.

Para instalar a CAPICOM, basta efetuar seu download (ver seção de links) e efetuar sua instalação. Depois de realizada a instalação da mesma, é registrada a TypeLibrary relacionada a ela, contudo, também pode ser registrada manualmente através do Prompt de comandos, dentro do diretório o sistema (system32), com a execução

Tipo	Descrição
A1	O processo de geração é baseado em Software. Este tipo de certificado pode ser criado e armazenado no computador do usuário. Neste caso, os dados são protegidos por uma senha e através da mesma, existe a possibilidade do acesso e da cópia da chave privada. Contudo, este tipo de certificado possui um nível menor de segurança.
A3	O processo de geração é baseado em Hardware. Este tipo de certificado é criado e gravado em um dispositivo criptográfico como um Token ou um cartão (Smart Card). As informações não podem ser copiadas, sendo protegidas por uma senha. Este tipo possui um nível de segurança médio / alto.

Tabela 1. Tipos de certificados digitais utilizados na NFe

Classe	Descrição
TStore	Provê métodos para o acesso ao Certificate Store do Windows. Através do método Open, é possível indicar qual o grupo do Store na qual se deseja ter acesso, por exemplo: 'My' para certificados pessoais, 'CA' para autoridades certificadoras e assim por diante.
TCertificate	É utilizado para a representação de um Certificado digital e seus atributos, como chave privada, chave pública, validade, entre outros.
TCertificates	Provê uma lista que contém os certificados de acordo com os parâmetros que foram definidos no momento em que o Store foi aberto, ou com base na função Select, utilizada para selecionar.

Tabela 2. Classes utilizadas para o acesso do Certificate Store através da CAPICOM

Classe	Descrição
TSigner	Armazena os dados do assinante do documento, bem como o certificado digital.
TSigners	Utilizada para armazenar uma coleção do tipo TSigner.
TSignedData	Permite que dados possam ser assinados, possibilitando também a verificação da assinatura em documentos já assinados.
TSignedCode	Provê a assinatura de determinado conteúdo com uma assinatura digital (AuthentiCode).

Tabela 3. Classes utilizadas para a assinatura de documentos com a CAPICOM

da linha de comando "regsvr32 capicom.dll". Concluído este processo, para efetuar sua importação no Delphi, basta utilizar a opção Import Component do Menu Component. Este assistente do Delphi permite a importação de controles ActiveX, biblioteca de tipos, entre outros, gerando uma Unit que pode ser gravada em conjunto com o projeto. Devido ao fato da CAPICOM ser uma biblioteca baseada em COM, é possível efetuar a extração de suas interfaces para uma Unit no Delphi através deste recurso.

Com o assistente aberto, basta manter a opção "Import a Type Library" selecionada e na próxima tela informar a opção "CAPICOM v2.1 Type Library". Na terceira etapa do assistente é interessante definir no campo "Unit Dir Name", o mesmo endereço onde estão os fontes do projeto para que Unit seja gravada em conjunto com os mesmos. Na tela final do assistente, basta utilizar a opção "Add Unit to..." para que a Unit seja adicionada ao projeto. Vale a pena evidenciar que a biblioteca CAPICOM deve posteriormente ser distribuída em conjunto com o Software, uma vez que a mesma está integrada. Após a conclusão do assistente de importação com as classes é gerada a unit CAPICOM_TLB.pas. As classes desta unit podem ser utilizadas para o acesso ao Certificate Store do Windows, assim como permitem a assinatura de documentos, a utilização de cripotografia, descriptografia de dados, entre outros recursos. Para acessar o Certificate Store, a CAPICOM possui diversas classes básicas. A **Tabela 2** demonstra as classes da CAPICOM utilizadas para o acesso ao Certificate Store do Windows.

Para efetuar uma assinatura digital através da biblioteca CAPICOM, existem algumas classes que podem ser utilizadas. A **Tabela 3** descreve algumas destas classes utilizadas para assinar documentos.

Prosseguindo com a elaboração do projeto, deve-se permitir que o usuário selecione um certificado, lembrando que a unit CAPICOM_TLB.pas deve ser adicionada à seção Uses. A **Listagem 1** ilustra como utilizar os recursos da CAPICOM para acessar o Certificate Store do Windows e selecionar um certificado.

A variável WindowsStore é instanciada através de seu devido construtor e posteriormente é utilizada para abrir os certifica-

Nota do DevMan

O COM (Component Object Model) é uma tecnologia independente que permite a criação de componentes de Software binários que podem interagir entre si, baseado em um sistema orientado a objetos. Através dele, os objetos podem ser reutilizados sem que suas implementações internas sejam conhecidas. Ele é a base para a utilização dos controles ActiveX e documentos OLE, entre outras tecnologias.

Nota do DevMan

A CryptoAPI é uma biblioteca que disponibiliza funções que possibilitam que aplicações possam assinar dados digitalmente ou criptografar informações. Suas operações de criptografia são realizadas através de módulos independentes, também conhecidos como CSPs (Cryptographic Service Providers).

Listagem 1. Procedure para selecionar um certificado digital

```
procedure TFormMovNotaFsical.SelecionarCertificado;
var
  WindowsStore: IStore3;
  Certitifacos: ICertificates2;
  Certificados2: ICertificates2;
  Certificado: ICertificate2;
begin
  try
    WindowsStore := CoStore.Create;
    WindowsStore.Open(CAPICOM_CURRENT_USER_STORE,'My',
    CAPICOM_STORE_OPEN_MAXIMUM_ALLOWED);
    Certitifacos := WindowsStore.Certificates as ICertificates2;
    Certificados2 := Certitifacos.Select('Certificados','Selecione o Certificado:', False);
    if Certificados2.Count > 0 then
    begin
      Certificado := IInterface(Certificados2.Item[1]) as ICertificate2;
      CertificadoAtual := Certificado;
    end;
  except
    on e: Exception do
      ShowMessage('Houve um erro: '+ e.Message);
  end;
end;
```

dos pessoais (My). Os certificados presentes em WindowsStore são atribuídos a variável denominada Certificados. Também é exibida a janela para que o usuário selecione o certificado desejado e para concluir, o mesmo é gravado na variável global CertificadoAtual. Selecionado o certificado, a próxima etapa é a assinatura da nota. Contudo, para efetuar a assinatura do documento XML, outro componente deverá ser obtido e importado para o Delphi através do mesmo ,assistente, porém, desta vez o objeto que deve ser selecionado é o "Microsoft XML, v5.0". O MSXML (Microsoft XML Parser) é um recurso que implementa o COM (Component Object Model) para a utilização de DOM (Document Object Model), disponível em diversas versões. Normalmente este componente já está presente no Windows pois é uma biblioteca compartilhada e utilizada por outros programas, mas em todo o caso também pode ser obtido através da Internet. A biblioteca pode ser encontrada com o nome de "msxml5.dll" (biblioteca MSXML versão 5.0), assim como a "msxml5r.dll" (biblioteca de recurso), que podem ser registradas através do comando regsvr32 através do Prompt. Ao concluir o assistente, será criada a unit MSXML2_TLB.pas que também pode ser gravada com o projeto. A **Listagem 2** demonstra uma função que pode ser utilizada para efetuar a assinatura do documento utilizando a CAPICOM em conjunto com a MSXML.

Na declaração das variáveis, alguns tipos utilizados são obtidos através das Units importadas das bibliotecas CAPICOM e MSXML e merecem ser destacados. O IXMLDOMDocument3 é utilizado para representar um documento XML, fornecendo diversos métodos para que o mesmo possa ser facilmente manipulado. O IXMLDigitalSignature é uma interface responsável pela assinatura digital de documentos baseados em XML, ela possui o método createKeyFromCSP, que posteriormente é utilizado neste exemplo, cujo o objetivo é recuperar uma chave contida em um fornecedor de serviços de criptografia (CSP), que posteriormente é utilizado para acessar a chave privada e assinar o documento. A interface IXMLDSigKey

Listagem 2. Função para assinar a Nota

```
function TFormMovNotaFsical.AssinarNFe(Certificado: ICertificate2;
  ArquivoXML: AnsiString): Boolean;
var
  PosId, PosSignature, i: Integer;
  Tipo, UriNFe: String;
  NFeAssinada: AnsiString;
  DocumentoDOM: IXMLDOMDocument3;
  AssinaturaXML: IXMLDigitalSignature;
  XMLAssina, XMLAssinada: IXMLDSigKey;
  MemoriaStore: IStore3;
  ChavePrivada: IPrivateKey;
  StrStream: TStringStream;
begin
  Tipo := 'NFe';
  try
    if Pos('<inf' + Tipo, ArquivoXML) > 0 then
    begin
      PosId := Pos('Id=',ArquivoXML);
      UriNFe := Copy(ArquivoXML, PosId + 4, 47);
      ArquivoXML := Copy(ArquivoXML, 1, Pos('</'+ Tipo +'>', ArquivoXML) -1);
      ArquivoXML := ArquivoXML + '<Signature xmlns=
      "http://www.w3.org/2000/09/xmldsig#">'+
                    '<SignedInfo><CanonicalizationMethod Algorithm='+
                    '"http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>'+
                    '<SignatureMethod Algorithm='+
                    '"http://www.w3.org/2000/09/xmldsig#rsa-sha1" />';
      ArquivoXML := ArquivoXML + '<Reference URI="#'+ UriNFe +'">
      <Transforms><Transform Algorithm='+
                    '"http://www.w3.org/2000/09/xmldsig#enveloped-signature" />'+
                    '<Transform Algorithm='+
                    '"http://www.w3.org/TR/2001/REC-xml-c14n-20010315" />
                   </Transforms>'+
                    '<DigestMethod Algorithm="http://www.w3.org/2000/09/
                       xmldsig#sha1" />';
      ArquivoXML := ArquivoXML + '<DigestValue></DigestValue></Reference>
      </SignedInfo>'+
                    '<SignatureValue></SignatureValue><KeyInfo></KeyInfo>
                     </Signature>';
      ArquivoXML := ArquivoXML + '</'+ Tipo +'>';
    end;
    DocumentoDOM := CoDOMDocument50.Create;
with DocumentoDOM do
    begin
      async := False;
      validateOnParse := False;
      preserveWhiteSpace := True;
      loadXML(ArquivoXML);
      setProperty('SelectionNamespaces', 'xmlns:ds="http://www.w3.org/2000/09/
      xmldsig#"');
    end;
    AssinaturaXML := CoMXDigitalSignature50.Create;
    AssinaturaXML.signature := DocumentoDOM.selectSingleNode('.//ds:Signature');
    MemoriaStore := CoStore.Create;
    MemoriaStore.Open(CAPICOM_MEMORY_STORE, 'Mem', CAPICOM_STORE_OPEN_
    MAXIMUM_ALLOWED);
    MemoriaStore.Add(Certificado);
    OleCheck(IDispatch(Certificado.PrivateKey).QueryInterface(IPrivateKey, ChavePrivada));
    AssinaturaXML.store := MemoriaStore;
    XMLAssina := AssinaturaXML.createKeyFromCSP(ChavePrivada.ProviderType,
                                 ChavePrivada.ProviderName,
                                 ChavePrivada.ContainerName, 0);
    XMLAssinada := AssinaturaXML.sign(XMLAssina, $00000002);
    NFeAssinada := DocumentoDOM.xml;
    NFeAssinada := StringReplace(NFeAssinada, Chr(13), '',[rfReplaceAll]);
    NFeAssinada := StringReplace(NFeAssinada, Chr(10), '',[rfReplaceAll]);
    PosSignature := Pos('<SignatureValue>', NFeAssinada) + 16;
    NFeAssinada := Copy(NFeAssinada, 1, PosSignature) +
              StringReplace(Copy(NFeAssinada, PosSignature, Length(NFeAssinada)),
                        '','',[rfReplaceAll]);
    StrStream := TStringStream.Create('');
    StrStream.WriteString(NFeAssinada);
    StrStream.SaveToFile('C:\FMXNFE\NfeAssinadas\'+ Copy(UriNFe,4,44) + '-nfe.xml');
    Result := True;
  except
    Result := False;
  end;
  XMLAssinada := nil;
  XMLAssina := nil;
  AssinaturaXML := nil;
  DocumentoDOM := nil;
end;
```

permite que uma aplicação possa acessar um certificado e efetuar chamadas aos objetos ou métodos da CAPICOM. Já o IStore3 é utilizado para fornecer métodos que permitem que os certificados possam ser selecionados e gerenciados, e para concluir, a IPrivateKey simplesmente é utilizada para representar uma chave privada. É feita uma pesquisa no arquivo XML através da função ""Pos" para verificar a existência do elemento <infNfe>. A função "Pos" retorna a posição de determinadao subtexto dentro ocorrência com relação ao de um texto, evidentemente se a mesmao existir. Posteriormente é obtida a posição inicial do atributo Id desta mesma Tag, e com base na mesma, é obtida a URI da NF-e, uma vez que tal informação também deve ser utilizada futuramente no processo da assinatura. Em seguida, é retirada a tag final da nota fiscal (</NFe>) para que seja inserido o elemento <Signature> em seus nós filhos. Posteriormente, o documentoDOM tem alguns de seus atributos definidos e através do método loadXML é carregado o XML da NF-e passada como parâmetro da função. A variável responsável por assinar o documento (AssinaturaXML) é inicializada e configurada a sua Signature. Na sequência é criado um IStore3 (MemoriaStore) para armazenar o Certificado atual selecionado. A função OleCheck é utilizada para rotinas COM, neste exemplo para a chave privada do certificado, e pode gerar uma exceção caso ocorra algum erro. Utilizando createKeyFromCSP é extraída a chave privada, tendo a realização da assinatura logo em seguida. A variável NFeAssinada recebe o XML do documento DOM assinado, removendo os possíveis espaços e quebra de linhas presentes na assinatura através da função "StringReplace", que permite a substituição de determinado caractere por outro, ou até mesmo sua remoção como neste caso. Finalmente é utilizada uma variável do tipo TStringStream para efetuar a gravação do documento assinado para o diretório especificado. O TStringStream é um objeto utilizado para armazenar e manipular caracteres, permitindo também o acesso a um disco para efetuar operações, como é o caso deste exemplo onde é realizada gravação do documento XML assinado em um diretório.

Nota do DevMan

O DOM (Document Object Model) é um recurso padronizado pela W3C que é utilizado para a representação e manipulação de documentos XML, entre outros, na memória, através de objetos. Utilizando o DOM, tarefas como alteração e exclusão de nós filhos do documento se tornam bem mais fáceis. Sendo assim, um documento XML como a NF-e pode ser mapeado para a memória, editado e posteriormente gravado com todas as alterações de uma única vez, simplificando o trabalho.

Nota do DevMan

A URI (Uniform Resourse Identifier) é em suma um endereço constituído de caracteres utilizado para identificar determinado recurso disponibilizado através da Internet. No caso da nota fiscal eletrônica, o URI é que deve ser utilizado é http://www.portalfiscal.inf.br/nfe.

Validação

Com a assinatura da nota efetuada, antes de ser transferida para a receita, convém efetuar uma validação utilizando seu respectivo Schema. Este processo é muito importante pois permite que sejam descobertos erros de estrutura ou dados inconsistentes presentes no documento XML. Sendo assim, se os mesmos são encontrados eles podem ser tratados antes que a NF-e seja enviada para a SEFAZ, e consequentemente seja rejeitada. Para efetuar a validação da mesma, basta efetuar o carregamento da nota na memória utilizando DOM e submetê-la a validação do Schema, uma vez que os recursos DOM implementados pela MSXML permitem esta validação de uma maneira simples. A **Listagem 3** demonstra uma função utilizando as classes da Unit MSXML2_TLP.pas para a validação de uma nota fiscal.

Listagem 3. Função para validar a Nota

```
function TFormMovNotaFsical.ValidarNFe(ArquivoXML: AnsiString; out Erro:
String): Boolean;
var
  DocumentoDOM: IXMLDOMDocument2;
  ErroEncontrado: IXMLDOMParseError;
  XMLSchema: IXMLDOMSchemaCollection2;
  ArquivoSchema, Tipo, UriNFe: String;
  PosId: Integer;
  StrStream: TStringStream;
begin
  Tipo := 'NFe';
  try
    if Pos('<inf' + Tipo, ArquivoXML) > 0 then
    begin
      DocumentoDOM := CoDOMDocument50.Create;
      with DocumentoDOM do
      begin
        async := False;
        resolveExternals := False;
        validateOnParse := True;
        loadXML(ArquivoXML);
      end;
      XMLSchema := CoXMLSchemaCache50.Create;
      ArquivoSchema := 'C:\FMXNFE\schemas\'+ Tipo +'_v2.00.xsd';
      XMLSchema.add('http://www.portalfiscal.inf.br/nfe', ArquivoSchema);
      DocumentoDOM.schemas := XMLSchema;
      ErroEncontrado := DocumentoDOM.validate;
      Erro := ErroEncontrado.reason;
      if ErroEncontrado.errorCode = 0 then
      begin
        PosId := Pos('Id=',ArquivoXML);
        UriNFe := Copy(ArquivoXML, PosId + 4, 47);
        StrStream := TStringStream.Create('');
        StrStream.WriteString(ArquivoXML);
        StrStream.SaveToFile('C:\FMXNFE\NfeValidadas\'+ Copy(UriNFe,4,44)
        + '-nfe.xml');
        Result := True;
      end
      else
        Result := False;
    end;
  finally
    XMLSchema := nil;
    DocumentoDOM := nil;
    ErroEncontrado := nil;
  end;
end;
```

Na declaração das variáveis deste exemplo, também existem alguns tipos que devem ser evidenciados. A interface IXMLDOMDocument2 é utilizada para a representação do documento XML, fornecendo também diversos métodos para a manipulação do mesmo. O IXMLDOMParseError é responsável por permitir a obtenção detalhada de possíveis erros de validação, como o número do erro, linha, posição, etc. Já a interface IXMLDOMSchemaCollection2 permite o carregamento de um schema XML. Após a declaração das variáveis, Éé verificado se a tag infNFe está presente no arquivo. Posteriormente o documento DOM é devidamente instanciado onde, na sequência, também é instanciado o XMLSchema, bem como são definidos o NameSpaceURI e o endereço do Schema. Após este processo, o documento é validado e seu retorno é atribuído à variável ErroEncontrado, onde o valor 0 representa a quantidade de erros. Para finalizar, é utilizada uma variável do tipo TStringStream (StrStream) para efetuar a gravação do arquivo validado em outra pasta do sistema.

Consumindo o WebService de Status

O WebService é um recurso muito importante utilizado pelas entidades que pode oferecer diversas funcionalidades através da Internet. Um WebService é um serviço automático que possui seus próprios métodos, onde o mesmo recebe uma requisição, processa e retorna o resultado para o solicitante. A Secretaria da Fazenda de cada estado disponibiliza diversos WebServices, cada um com uma finalidade, onde a aplicação cliente envia uma mensagem para o WebService desejado para obter o serviço necessário. Os WebServices da SEFAZ sempre retornam uma mensagem informando o recebimento na mesma conexão. WebServices em si, podem oferecer serviços síncronos ou assíncronos. Nos serviços síncronos, como é o caso da consulta de status, o processamento da requisição efetuada pela aplicação cliente é concluída na mesma conexão. Já nos serviços assíncronos, como o processo de autorização da NF-e, a aplicação cliente recebe uma resposta que confirma o recebimento, porém, a mesma deve efetuar uma ou mais consultas posteriormente (uma nova conexão) para obter a informação desejada.

Os WebServices da NF-e são padronizados em documentos XML, utilizando UTF-8. É importante ressaltar que os documentos enviados a receita devem conter apenas uma declaração "<?xml version="1.0" encoding="UTF-8"?>".

A comunicação com os WebServices é feita utilizando o protocolo SSL, tendo autenticação mútua, permitindo uma maior segurança das informações, sendo assim, além de utilizado na geração da assinatura da NF-e, o certificado digital também é necessário neste momento, como foi mencionado anteriormente. As mensagens transmitidas entre a aplicação cliente e os WebServices também é baseada no protocolo SOAP versão 1.2, devendo conter um cabeçalho (elemento Header) e obrigatoriamente o corpo (elemento Body).

A consulta ao WebService de Status é apresentada utilizando um objeto HTTPReqResp.Este componente que está presente por padrão no Delphi, utiliza o protocolo HTTP (HyperText Transfer Protocol) para efetuar requisições no padrão SOAP para um Web Service, obtendo também uma mensagem de retorno do mesmo, sendo assim, ao conhecer o funcionamento do WebService, ou em outras palavras, as informações que devem estar presentes na mensagem SOAP, o desenvolvedor pode utilizar o mesmo para o consumo. A **Listagem 4** demonstra o código utilizado para consumir o WebService de Status utilizando um objeto HTTPReqResp em conjunto com uma mensagem SOAP, já obtendo o valor de retorno.

O objeto HTTPReqResp é criado e suas propriedades são definidas de acordo com a necessidade do WebService. A mensagem em padrão SOAP é atribuída à variável MsgSOAP para uma pesquisa de status do WebService de SP (35). Também é criado um TMemoryStream (variável MemoriaStream) para armazenar os dados retornados na memória. A variável TStringStream copia os dados da variável MemoriaStream e obtém o valor em String. Concluindo, a String de retorno é formatada e posteriormente filtrada.

Nota do DevMan

O XML (Extensible Markup Language) é um documento baseado em uma linguagem de marcação, padronizado pela W3C que pode ser utilizado para o armazenamento de dados. É um dos principais meios adotado para a troca de diversas informações através da Internet, como é o caso da NFe e o consumo de WebServices. Por ser extensível, não existe um limite para a quantidade de Tags que um documento XML pode ter, uma vez que as mesmas podem ser criadas pelo próprio desenvolvedor de acordo com sua necessidade. Quanto a sua constituição interna, o documento XML é composto por elementos ou "nós" que armazenam as informações obedecendo a uma determinada Hierarquia.

Nota do DevMan

A W3C (World Wide Web Consortium) é um consórcio internacional constituído por várias instituições que tem por fim o estabelecimento de padrões para a elaboração e interpretação de diversos documentos presentes na Web. Estes padrões visão permitir que qualquer aplicação, independente de plataforma, possa utilizar os recursos da Internet para a troca de informações.

Nota do DevMan

O protocolo SOAP (Simple Object Access Protocol) é um padrão que foi desenvolvido pela W3C sendo baseado em XML (Extensible Markup Language) que foi elaborado com o objetivo de ser a base para as chamadas de aplicações remotas utilizando o RPC (Remote Procedure Calls), permitindo assim a troca de informações entre as entidades envolvidas. O SOAP possui várias versões e é utilizado em conjunto com o protocolo HTTP, que por suas vez possibilita o transporte das informações.

Nota do DevMan

O protocolo HTTP (Hypertext Transfer Protocol) é o padrão utilizado atualmente na Web para a troca de informações entre a aplicação servidor e a cliente, efetuando o tratamento do pedido e da resposta. Quando a abertura de um site da Web é solicitada através um navegador, o HTTP é o protocolo adotado para esta tarefa.

Listagem 4. Função para consumir o WebService de Status da SEFAZ (Homologação)

```
function TFormMovNotaFsical.ConsultarStatusServico(
 out Mensagem: string): Boolean;
var
  MemoriaStream: TMemoryStream;
  TextoStream: TStringStream;
  MsgSOAP, Resultado: String;
  Lista: TStringList;
  HTTPReqResp: THTTPReqResp;
begin
  HTTPReqResp := THTTPReqResp.Create(FormMovNotaFsical);
  try
    with HTTPReqResp do
    begin
      ClientCertificate.SerialNum := CertificadoAtual.SerialNumber;
      InvokeOptions := [soIgnoreInvalidCerts,soNoSOAPActionHeader,
                soPickFirstClientCertificate,soAutoCheckAccessPointViaUDDI];
      URL := 'https://homologacao.nfe.fazenda.sp.gov.br/nfeweb/services/
      NfeStatusServico2.asmx';
      UseUTF8InHeader := False;
      SoapAction := 'http://www.portalfiscal.inf.br/nfe/wsdl/NfeStatusServico2';
    end;
    Lista := TStringList.Create;
    MsgSOAP := '<?xml version="1.0" encoding="utf-8"?>'+
          '<soap12:Envelope xmlns:xsi="http://www.w3.org/2001/
           XMLSchema-instance"'+
          'xmlns:xsd="http://www.w3.org/2001/XMLSchema"'+
          'xmlns:soap12="http://www.w3.org/2003/05/soap-envelope">'+
          '<soap12:Header>'+
          '<nfeCabecMsg xmlns="http://www.portalfiscal.inf.br/nfe/wsdl/
            NfeStatusServico2">'+
'<cUF>35</cUF>'+
          '<versaoDados>2.00</versaoDados>'+
          '</nfeCabecMsg>'+
          '</soap12:Header>'+
          '<soap12:Body>'+
          '<nfeDadosMsg xmlns="http://www.portalfiscal.inf.br/nfe/wsdl/
            NfeStatusServico2">'+
          '<consStatServ xmlns="http://www.portalfiscal.inf.br/nfe" versao="2.00">'+
          '<tpAmb>2</tpAmb>'+
          '<cUF>35</cUF>'+
          '<xServ>STATUS</xServ>'+
          '</consStatServ>'+
          '</nfeDadosMsg>'+
          '</soap12:Body>'+
          '</soap12:Envelope>';
    Lista.Text := MsgSOAP;
    MemoriaStream := TMemoryStream.Create;
    HTTPReqResp.Execute(Lista.Text, MemoriaStream);
    TextoStream := TStringStream.Create('');
    TextoStream.CopyFrom(MemoriaStream, 0);
    Resultado := TextoStream.DataString;
    Resultado := UTF8Decode(Resultado);
    Resultado := Copy(Resultado, Pos('<xMotivo', Resultado) + 9,
              Pos('</xMotivo', Resultado) - (Pos('<xMotivo', Resultado) + 9));
    ShowMessage(Resultado);
  finally
    FreeAndNil(HTTPReqResp);
  end;
end;
```

Conclusão

A biblioteca CAPICOM é um recurso muito importante na elaboração de um projeto de aplicação que suporte a Nota Fiscal Eletrônica, oferecendo uma maior facilidade para utilizar os certificados digitais do Windows, bem como as funções necessárias para a assinatura do documento XML. Vale a pena evidenciar que este é um componente gratuito, disponibilizado pela Microsoft. Contudo, também existem outras bibliotecas alternativas que podem ser utilizadas, como é o caso da OpenSSL.

A biblioteca MSXML também é uma ferramenta muito útil e gratuita para ser utilizada no projeto de aplicação NF-e, uma vez que provê métodos para o mapeamento de documentos XML utilizando os recursos DOM.

Como mostrado, o WebService de Status da SEFAZ pode ser facilmente consumido através de um objeto do tipo THTTPReqResp, utilizando uma mensagem no padrão SOAP de acordo com as especificações da mesma.

No próximo artigo serão abordados os temas de envio da NF-e, sua autorização, bem como outros recursos.

Giuliano Scombatti Pinto
giuliano@sygnux.com.br
É Analista Desenvolvedor da Sygnux Software (www.sygnux.com.br) localizada em Monte Alto/SP, atuando também com desenvolvimento Web e Android. Formado pela Fatec/TQ, começou a desenvolver com Delphi em 2003. Atualmente trabalha com Delphi, Java, PHP e Flash/Flex. Professor de Informática de Projetos Sociais da Prefeitura Municipal de Monte Alto/SP.

Download do Manual do Contribuinte e dos Schemas
https://www.fazenda.sp.gov.br/nfe/downloads/arquivos_vigentes.asp

Download do CAPICOM
http://www.microsoft.com/downloads/details.aspx?FamilyID=860ee43a-a843-462f-abb5-ff88ea5896f6&displaylang=pt-br

DevSpace do Autor
http://www.devmedia.com.br/space.asp?id=283274

Dê seu feedback sobre esta edição!

A Clubedelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre este artigo, através do link:

www.devmedia.com.br/clubedelphi/feedback

Desmistificando o Exe
Executando instruções PSQL for

Resumo DevMan

De que trata o artigo:

O artigo trata de ilustrar o Execute Block, um recurso específico do Firebird que assim como Procedures, permite a utilização da linguagem PSQL para a criação de procedimentos mais elaborados de forma dinâmica, sem a necessidade de alocar o objeto no banco de dados.

Em que situação o tema é útil:

O tema é útil em casos onde é necessário realizar alterações a partir de seleções complexas, o que demandaria a criação de uma Procedure, trazendo consigo a vantagem de executar qualquer comando presente em PSQL de forma simplificada, sem a necessidade de ser efetivamente criado no banco de dados.

Desmistificando o Execute Block:

Em situações de atualizações de Software, ou mesmo procedimentos que envolvem grandes quantidades de alterações, sempre é comum os desenvolvedores ou administradores do banco de dados utilizarem recursos da linguagem PSQL para criar Procedures que permitam operações de manipulação de várias tabelas, inserções, atualizações, alterações, adaptações entre outras diversas situações. Em muitos destes casos, estas Procedures são criadas unicamente para realização destes procedimentos, porém, esquecidas, acabam se tornando apenas objetos inúteis no banco de dados. O objetivo é demonstrar um recurso presente desde a versão 2.0 do Firebird, na qual permite a execução de blocos de PSQL de maneira dinâmica, de forma desacoplada do banco de dados, ilustrando além de sua finalidade as formas e maneiras de como utilizá-lo em diversas ocasiões do cotidiano.

Utilizar recursos do banco de dados é algo que sempre rende boas discussões entre desenvolvedores. Há quem diga que o banco de dados deve funcionar como mero repositório de dados, assim como há quem diga que se deve aproveitar ao máximo os recursos disponíveis em um SGDB. É claro que esta questão possui prós e contras. Uma dessas vantagens de se utilizar os recursos de banco de dados é a autonomia para a realização de procedimentos automáticos de maneira transparente à aplicação, como baixa de estoque, geração de contas a pagar/receber, geração de duplicatas entre muitas outras coisas. Práticas comuns do cotidiano, muitas destas rotinas citadas são utilizadas em forma de Procedures no banco de dados.

Em uma situação real, de uma venda, pode-se realizar a baixa do estoque dos produtos selecionados, criar um lançamento no contas a receber (quitando ou não o lançamento), emitir a nota fiscal, lançar a comissão do vendedor, entre tantas outras funcionalidades comuns a um Software comercial. Como tudo há vantagens e desvantagens, o lado negativo é que para isso, há a necessidade de se criar alguns conjuntos de objetos específicos para essas ações, como as Procedures, já mencionadas, e Triggers para validações necessárias, e isto, pode fazer com que a aplicação fique muito dependente do banco de dados. Embora isto não seja uma regra, uma boa prática é sempre avaliar o custo/benefício de se criar os recursos em aplicação ou banco de dados, procurando

cute Block
a do banco de dados

Daniel Sobrinho Laporte

manter um equilíbrio entre ambos e não tornando a aplicação totalmente dependente do banco de dados, facilitando em caso de uma futura migração. Além destas questões já comentadas, deve-se avaliar também o ambiente. O excesso de objetos (Triggers em particular) pode causar certa redução de desempenho, o que em caso de uma ambiente Web, torna-se completamente inviável.

Embora haja inúmeras questões que podem ser levadas em conta, um dos grandes problemas em Software Houses são as atualizações. Independente de se utilizar as regras concentradas em aplicação, não há como escapar das alterações em DDL. Como se sabe, se o volume de alterações no banco de dados é grande, logicamente haverá uma grande quantidade de Scripts a serem executados no momento da atualização, o que pode demandar um pouco mais de trabalho por parte da equipe de suporte, ou do próprio Software se o mesmo se "auto-atualiza". Independente de adotar a metodologia de centralização das regras no banco de dados ou não, uma coisa é certa, não há como fugir de alterações no SGDB.

Um exemplo claro destas dificuldades mais do que nunca está presente agora. Com as recentes exigências do governo em relação à fiscalização (SPED, NF-e, CT-e entre tantos outros), as Software Houses estão adequando seus produtos as conformidades estabelecidas pelo governo. E nestes casos, as alterações no banco de dados em parte estrutural (DDL) ou mesmo em dados já consistentes, tornaram-se tarefa comum aos desenvolvedores, administradores ou DBAs em geral.

Em se tratando deste tipo de atualização, comumente quando há grandes alterações estruturais que envolvem criação de várias tabelas, importação de dados através de arquivos, planilhas ou algo do tipo, são criados procedimentos no banco de dados que realizam este tipo de tarefa, buscando informações de diversas tabelas para "compor" outras, validando dados e os persistindo em seguida, criações de tabelas temporárias (Global Temporary Tables) ou auxiliares, entre tantas outras necessidades. Neste tipo de situação as únicas maneiras de se trabalhar com várias tabelas simultaneamente, eram desenvolver pequenos aplicativos que realizassem o trabalho de conexão com o banco de dados, seleção de tabelas, manuseio dos dados e etc, apenas para realizar estes procedimentos. Ou utilizando a segunda opção, na qual era trabalhar com Procedures criadas unicamente para estes fins, que executassem estas tarefas e fizessem o trabalho árduo, sendo logo em seguida excluídas, ou, em muitos casos, nem mesmo isso, tornando-se apenas objetos inúteis e contribuindo com o crescimento excessivo do banco de dados de forma errada.

Nota do DevMan

Global Temporary Tables assim como tabelas comuns, permitem o manuseio de dados normalmente. Possuem algumas particularidades como não permitir o uso de domínios em sua definição, além de permitir estabelecer transações independentes das demais concorrentes no banco de dados. Estas transações são encerradas ao final da utilização do recurso ou no momento da desconexão, porém, rodam isoladamente.

As tabelas temporárias estão presentes em outros bancos de dados além do Firebird, assim como Oracle por exemplo. No Firebird este recurso está presente a partir da versão 2.1. A sintaxe para a criação de uma Global Temporary Table é:

```
CREATE GLOBAL TEMPORARY TABLE NOME_TABELA
( CAMPOS TIPO_DE_DADOS );
```

Até a versão 1.5 do Firebird a única maneira de realizar este tipo de atualização era por meio de Procedures e, embora, o Firebird esteja na versão 2.5, cheio de recursos no mínimo interessantes, pouco se fala destes recursos, ou mesmo, pouco se usa, porém, a partir da versão 2.0 já é possível realizar estas atualizações mencionadas de maneira mais simples. Isto porque esta versão (que já é antiga) trouxe consigo um recurso muito interessante para auxiliar na criação e manuseio de objetos complexos. Este recurso é denominado de Execute Block. Com ele é possível criar objetos definidos a partir da linguagem de PL/SQL ou PSQL como é chamada no Firebird. A linguagem PSQL é a mesma utilizada em Procedures e Triggers e permitem realizar alterações de estrutura (Data Definition Language - DDL) ou alteração de dados (Data Manipulation Language – DML) com facilidade e muita flexibilidade. Pouco diferente, o Execute Block faz uso de uma linguagem específica chamada de DSQL (Dynamic Structured Query Language) que seria idêntica ao PSQL se não fosse o grande diferencial deste recurso: o Execute Block não fica armazenado como um objeto fixo do banco de dados. Em outras palavras, diferentemente dos objetos de banco de dados que utilizam a PSQL, o Execute Block pode ser executado em qualquer momento como se fosse uma chamada comum à uma tabela ou procedure já criada. Isto é um grande diferencial, pois, evita com que o banco de dados possua inúmeros objetos que serão pouco utilizados (caso da atualização), ou mesmo para a criação de relatórios ou rotinas mais complexas, reduzindo o crescimento do banco de dados.

Logicamente, embora seja um recurso dinâmico, as Procedures possuem mais performance no processo de execução, isso porque já estão pré-compiladas no banco de dados no momento em que estão disponíveis para uso. No caso do Execute Block, as operações

que são realizadas pelo validador do banco de dados que faz todo o processo de validação de sintaxe e pré-compilação, para só então executar as instruções.

Assim como o caso das Procedures, o Execute Block permite parâmetros de entrada e saída, ou seja, seu funcionamento é semelhante ao de uma Stored Procedure, permitindo assim realizar procedimentos selecionáveis ou somente de execução. Outra utilização alternativa e interessante para o Execute Block é o Debug. Ao permitir execução de blocos de código retornando valores, faz-se possível extrair pequenos blocos de código de outras Procedures, transformando-os em pequenos Execute Blocks e assim, resolvendo os possíveis problemas, ou mesmo criar procedimentos aleatórios para alimentar tabelas de teste (selecionando de dados de várias tabelas e persistindo em outras).

Nota do DevMan

PL/SQL é a abreviação de Procedural Language SQL. Presente em vários bancos de dados é uma extensão da linguagem SQL de forma poderosa, na qual permite a manipulação de informações relativas à estrutura ou mesmo manipulação completa de dados. Possui uma sintaxe semelhante ao SQL. No Firebird é conhecida apenas como PSQL. Objetos como Procedures e Triggers fazem uso desta linguagem. O Execute Block, assunto deste artigo, utiliza esta linguagem de forma dinâmica, desacoplada do banco de dados, deste modo, é conhecida como DSQL (Dynamic SQL).

Conhecendo a sintaxe

Como já mencionado, o Execute Block possui uma sintaxe muito semelhante à de uma Stored Procedure comum (devido à utilização do PSQL). Nota-se um exemplo de sintaxe de elaboração de uma Procedure a seguir:

```
CREATE PROCEDURE PROCEDURE_NAME (PARAMETROS_DE_ENTRADA TIPO_DE_DADOS)
RETURNS (PARAMETROS_DE_SAIDA TIPO_DE_DADOS)
AS
BEGIN
...
END;
```

Devido ao fato do Firebird não possuir Functions, quando tratar-se de uma Procedure selecionável os parâmetros de retorno devem ser obrigatoriamente informados. Nestes casos, o que fará com que os dados sejam retornados é a cláusula SUSPEND. Na declaração da sintaxe do Execute Block é claramente perceptível a semelhança entre ambos:

```
EXECUTE BLOCK (PARAMETROS_DE_ENTRADA TIPO_DE_DADOS = VALOR_PARAMETRO)
RETURNS (PARAMETROS_DE_SAIDA TIPO_DE_DADOS)
AS
BEGIN
...
END;
```

Como nota-se, a sintaxe de ambas as estruturas são muito semelhantes. Criar estruturas de Execute Block a partir de Procedures é muito simples, alterando basicamente os cabeçalhos de um do outro. Um exemplo simples é ilustrado na **Listagem 1,** propondo o caso já citado anteriormente aonde é mostrado o produto de uma multiplicação a partir de dois números distintos nas formas de Procedure e Execute Block:

Listagem 1. Comparando códigos de uma Procedure e um Execute Block

```
/*PROCEDURE*/
CREATE OR ALTER PROCEDURE MULTIPLICACAO(VALOR1 INTEGER, VALOR2
INTEGER)
RETURNS
 (
  RESULTADO INTEGER
 )
AS
BEGIN
 RESULTADO = COALESCE(VALOR1,0) * COALESCE(VALOR2,0);
 SUSPEND;
END;

/*EXECUTE BLOCK*/
EXECUTE BLOCK(VALOR1 INTEGER = :PARAMETRO1, VALOR2 INTEGER = :PARA-
METRO2)
RETURNS
 (
  RESULTADO INTEGER
 )
AS
BEGIN
 RESULTADO = COALESCE(VALOR1,0) * COALESCE(VALOR2,0);
 SUSPEND;
END;
```

O modo de criação de um Execute Block é o mesmo de uma Procedure, diferenciando apenas no cabeçalho (primeira linha), porém, o processo de execução é diferenciado. A chamada desta Procedure é realizada da seguinte maneira:

```
SELECT
 MULTIPLICACAO.RESULTADO
FROM
 MULTIPLICACAO(:VALOR1, :VALOR2)
```

No caso do Execute Block este processo ocorre diretamente, ou seja, executando o código da **Listagem 1** informando os parâmetros VALOR1=2 e VALOR2=10, o retorno RESULTADO exibirá o valor 20.

Nota

Na Procedure MULTIPLICACAO alguns detalhes chamam atenção e devem ser levados em conta. O primeiro diz respeito a utilizar apenas a cláusula CREATE em Procedures. Isto pode ser arriscado em caso de atualizações, pois, caso esta Procedure já exista dentro do banco de dados, o Script Executor não irá prever este problema, tentando criá-la novamente e proporcionando assim erros desnecessários. O ideal neste caso é utilizar CREATE OR ALTER, pois assim, não existindo, esta será criada e caso contrário, a mesma será atualizada. O segundo detalhe diz respeito ao COALESCE. Nos parâmetros VALOR1 e VALOR2 podem ser informados valores nulos, o que obviamente resultaria em uma multiplicação inválida ou pior, uma mensagem de erro. O COALESCE garante que os valores nulos sejam substituídos por zero (0), solucionando este possível problema.

Nota do DevMan

Assim como no Delphi, alguns bancos de dados como Oracle ou SQL Server, por exemplo, possuem suporte ao recurso de Functions que é utilizado quando deseja se obter algum tipo de informação, como o produto da multiplicação de dois números distintos. Em Firebird este recurso é simulado por meio das próprias Procedures que são selecionáveis ou não. Neste caso, diferem pelos parâmetros de Output (retornos). No Firebird é possível utilizar um exemplo de Procedure deste como uma SQL comum. Exemplo:

Select

Campos

From

Nome_Procedure(Parâmetros de entrada)

Where

Parâmedtros de saída = condições

Em casos de procedures que apenas executam algum tipo de atualização é utilizado o comando Execute. Exemplo:

Execute Procedure Nome_Procedure(Parâmetros de entrada);

Nota do DevMan

A RTTI (Run-Time Type Information) é uma técnica que permite se obter vários tipos de informação a respeito de objetos e classes em tempo de execução, fazendo assim com que seja possível mapear métodos, realizar checagens e Casting de classes, entre outras situações. São constantemente utilizadas em Frameworks ORM para obter informações sobre atributos de classes, auxiliando no processo de mapeamento, ou em técnicas mais avançadas e amplamente utilizadas na orientação a objetos e tráfego de objetos, como o processo de serialização de objetos por exemplo.

Execute Block e tabelas de sistema

No Firebird, utilizar tabelas de sistema pode proporcionar a criação de códigos mais dinâmicos, permitindo assim uma manipulação melhor dos recursos do sistema ou mesmo flexibilidade na criação de novos recursos. É claro que fazer uso das tabelas do sistema exige um conhecimento um pouco mais avançado, porém, com um pouco de criatividade, é possível usá-las para criação de códigos no mínimo interessantes. Assim como em Procedures, utilizar tabelas de sistema com Execute Block é perfeitamente possível e até mesmo simples. Citando um destes casos, a **Listagem 2** ilustra uma forma de simulação que pode ser utilizada como uma alternativa ou algo semelhante em vários tipos de sistemas. A suíte de componentes User Control, muito conhecida pelos desenvolvedores Delphi de longa data, faz uso de recursos de RTTI para mapeamento dos formulários do projeto, posteriormente, os salvando em bancos de dados. Em uma situação semelhante, a Procedure utiliza as tabela de sistema RDB$RELATIONS para mapeamento das demais tabelas. No exemplo são ignoradas as tabelas que iniciam com as siglas MON$ e RDB$ por tratar-se de tabelas do sistema, além da própria tabela que também é ignorada (caso já exista). Este procedimento poderia ser executado logo no início da aplicação, verificando a criação de novas tabelas e já permitindo ou não seu acesso. No caso da **Listagem 2** é ilustrado este exemplo por meio tanto da Procedure quanto do Execute Block. A única diferença no caso é exatamente a primeira linha, já que ambas não possuem parâmetros de entrada e saída. Novamente vale reforçar que este procedimento é dinâmico, portanto, caso seja executado no SQL Editor, por exemplo, o resultado será imediato.

Listagem 2. Procedure de mapeamento de tabelas

```
/*
PROCEDURE
*/
CREATE OR ALTER PROCEDURE MAPEAMENTO_DB
AS
DECLARE VARIABLE V_NOME_TABELA VARCHAR(50);
BEGIN
/*
 PRESSUPÕE QUE JÁ EXISTE A TABELA COM A SEGUINTE DDL:
 CREATE TABLE TABELA (
  IDTABELA    INTEGER NOT NULL PRIMARY KEY,
  NOME_TABELA  VARCHAR(50) NOT NULL,
  VISIVEL     CHAR(1));
*/
 --MAPEIA TODAS AS TABELAS DIFERENTES DA PRÓPRIA TABELA E
 --DIFERENTES DAS TABELAS DE SISTEMA
 FOR SELECT
  RDB$RELATIONS.RDB$RELATION_NAME
 FROM
  RDB$RELATIONS
 WHERE
  RDB$RELATIONS.RDB$RELATION_NAME NOT STARTING WITH 'MON$' AND
  RDB$RELATIONS.RDB$RELATION_NAME NOT STARTING WITH 'RDB$' AND
  RDB$RELATIONS.RDB$RELATION_NAME <> 'TABELA'
 INTO
  :V_NOME_TABELA
 DO
 BEGIN
  --SE NÃO ENCONTRA O REGISTRO DA TABELA, ENTÃO ELE É CRIADO
  IF (NOT EXISTS(SELECT NOME_TABELA FROM TABELA WHERE
          NOME_TABELA = :V_NOME_TABELA)) THEN
   INSERT INTO TABELA
    (
     IDTABELA,
     NOME_TABELA,
     VISIVEL
    )
   VALUES
    (
     (SELECT COALESCE(MAX(IDTABELA),0) + 1 FROM TABELA),
     :V_NOME_TABELA,
     'S'
    );
 END
END


/*
EXECUTE BLOCK
*/
EXECUTE BLOCK --ÚNICA MUDANÇA É REALIZADA NESTA LINHA
AS
DECLARE VARIABLE V_NOME_TABELA VARCHAR(50);
BEGIN
/*
 PRESSUPÕE QUE JÁ EXISTE A TABELA COM A SEGUINTE DDL:
 CREATE TABLE TABELA (
  IDTABELA    INTEGER NOT NULL PRIMARY KEY,
```

Continuação: Listagem 2. Procedure de mapeamento de tabelas

```
  NOME_TABELA  VARCHAR(50) NOT NULL,
  VISIVEL    CHAR(1));
*/
 FOR SELECT
  RDB$RELATIONS.RDB$RELATION_NAME
 FROM
  RDB$RELATIONS
 WHERE
  RDB$RELATIONS.RDB$RELATION_NAME NOT STARTING WITH 'MON$' AND
  RDB$RELATIONS.RDB$RELATION_NAME NOT STARTING WITH 'RDB$' AND
  RDB$RELATIONS.RDB$RELATION_NAME <> 'TABELA'
 INTO
  :V_NOME_TABELA
 DO
 BEGIN
  IF (NOT EXISTS(SELECT NOME_TABELA FROM TABELA WHERE
          NOME_TABELA = :V_NOME_TABELA)) THEN
  INSERT INTO TABELA
   (
    IDTABELA,
    NOME_TABELA,
    VISIVEL
   )
  VALUES
   (
    (SELECT COALESCE(MAX(IDTABELA),0) + 1 FROM TABELA),
    :V_NOME_TABELA,
    'S'
   );
 END
END
```

Criando um banco de dados de exemplo

Para complementar os exemplos demonstrados até aqui é criado um banco de dados de exemplo no qual sugere uma modelagem simples, porém, mais próxima de uma situação real de um banco de dados qualquer. Para isso são utilizadas quatro tabelas, sendo 3 relacionadas, além da tabela mencionada no exemplo da **Listagem 2.** O objetivo é ilustrar o funcionamento tanto da Procedure anterior, quanto do Execute Block, exibindo o funcionamento de maneira idêntica, além de ilustrar uma situação onde haja a necessidade de realizar uma alteração "complexa" na qual é feita a modificação na estrutura de uma tabela juntamente com a atualização de dados.

A **Figura 1** ilustra as entidades Cliente, Endereco e Estado que formam os relacionamentos até o momento. Na mesma figura é exibida a tabela "TABELA" que não possui relacionamento algum, porém, simulando o caso do mapeamento das demais tabelas. A **Listagem 3** contém juntamente com a criação do banco de dados, algumas instruções para a alimentação do banco de dados, no intuito de facilitar a manipulação de informações.

Nota

Para utilizar os recursos deste artigo, deve-se possuir obrigatoriamente ao menos a versão 2.0 do Firebird instalada, caso contrário, os recursos do Execute Block não funcionarão. Já para utilizar o recurso da Global Temporary Table, deve-se possuir ao menos a versão 2.1.

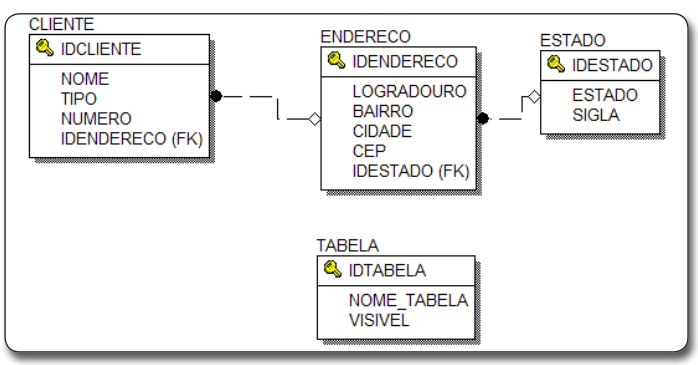


Figura 1. Modelagem proposta do artigo

Nela foram inseridos alguns dados automaticamente nas tabelas. Suponha-se em uma situação qualquer que está é a estrutura original do banco de dados, porém, para efeitos de melhoria, o objetivo seja dividir os clientes do tipo 'F' (pessoas físicas) dos 'J' (pessoas jurídicas). Neste caso, seriam criadas as tabelas PESSOA_FISICA e PESSOA_JURIDICA, fazendo posteriormente o relacionamento dos dados da tabela cliente entre estas tabelas e, posteriormente, excluindo o campo TIPO da tabela CLIENTE, que nesse caso se tornaria desnecessário. Isto é feito no mesmo Execute Block, porém, em duas etapas. Isto porque, caso sejam feitas as alterações de criação e inserção dos dados na nova tabela, elas estariam participando da mesma transação, o que não seria possível, pois, os dados relativos à criação das tabelas ainda não estariam "comitados" no momento da inserção. O código do Execute Block que faz a criação das devidas encontra-se a seguir:

```
EXECUTE BLOCK
AS
BEGIN
 EXECUTE STATEMENT 'CREATE TABLE PESSOA_FISICA(
          IDPESSOA_FISICA INTEGER NOT NULL PRIMARY KEY ,
          IDCLIENTE INTEGER NOT NULL);';

 EXECUTE STATEMENT 'CREATE TABLE PESSOA_JURIDICA(
          IDPESSOA_JURIDICA INTEGER NOT NULL PRIMARY KEY ,
          IDCLIENTE INTEGER NOT NULL);';
END
```

Executando a parte anterior, as tabelas são criadas. No caso do código anterior, não há relacionamento de chaves estrangeiras. Isso devido ao mesmo problema transacional, uma saída para isto é vista na **Listagem 4**, com a modificação para que os dados sejam separados nas novas tabelas.

O Execute Block foi alterado com a inserção de um parâmetro. Este parâmetro determina o tipo de ação na qual fará as etapas. A primeira etapa é dada com o preenchimento do parâmetro utilizando o caractere 'C' para o modo de criação. A segunda é realizada a partir do caractere 'I' que indicará o processo de inserção. Uma situação ainda poderia ser utilizada através do

Listagem 3. Criação do banco de dados

```
SET SQL DIALECT 3;

SET NAMES WIN1252;

SET CLIENTLIB 'fbclient.dll';

CREATE DATABASE '127.0.0.1:C:\CLUBEDELPHI\CD139.fdb'
USER 'SYSDBA' PASSWORD 'masterkey'
PAGE_SIZE 8192
DEFAULT CHARACTER SET WIN1252 COLLATION WIN1252;

CREATE TABLE CLIENTE (
  IDCLIENTE  INTEGER NOT NULL,
  NOME       VARCHAR(250),
  TIPO       CHAR(1),
  NUMERO     INTEGER,
  IDENDERECO  INTEGER
);

CREATE TABLE ENDERECO (
  IDENDERECO  INTEGER NOT NULL,
  LOGRADOURO  VARCHAR(250),
  BAIRRO     VARCHAR(250),
  CIDADE     VARCHAR(250),
  CEP        VARCHAR(8),
  IDESTADO   INTEGER NOT NULL
);

CREATE TABLE ESTADO (
  IDESTADO  INTEGER NOT NULL,
  ESTADO    VARCHAR(250) NOT NULL,
  SIGLA     CHAR(2)
);

ALTER TABLE CLIENTE
ADD CONSTRAINT PK_CLIENTE PRIMARY KEY (IDCLIENTE)
USING INDEX IDX_CLIENTE;

ALTER TABLE ENDERECO
ADD CONSTRAINT PK_ENDERECO PRIMARY KEY (IDENDERECO)
USING INDEX IDX_ENDERECO;

ALTER TABLE ESTADO
ADD CONSTRAINT PK_ESTADO PRIMARY KEY (IDESTADO)
USING INDEX IDX_ESTADO;

ALTER TABLE CLIENTE
ADD CONSTRAINT FK_CLIENTE_ENDERECO
FOREIGN KEY (IDENDERECO) REFERENCES ENDERECO (IDENDERECO)
ON DELETE CASCADE
ON UPDATE CASCADE
USING INDEX IDX_CLIENTE_ENDERECO;

ALTER TABLE ENDERECO
ADD CONSTRAINT FK_ENDERECO_ESTADO
FOREIGN KEY (IDESTADO) REFERENCES ESTADO (IDESTADO)
ON DELETE CASCADE
ON UPDATE CASCADE
USING INDEX IDX_ENDERECO_ESTADO;
CREATE TABLE TABELA (
IDTABELA    INTEGER NOT NULL PRIMARY KEY,
NOME_TABELA  VARCHAR(50) NOT NULL,
VISIVEL     CHAR(1));

/*ESTADOS*/
INSERT INTO ESTADO (IDESTADO, ESTADO, SIGLA) VALUES (1, 'ACRE', 'AC');
INSERT INTO ESTADO (IDESTADO, ESTADO, SIGLA) VALUES (2, 'ALAGOAS', 'AL');
INSERT INTO ESTADO (IDESTADO, ESTADO, SIGLA) VALUES (3, 'AMAPA', 'AP');
INSERT INTO ESTADO (IDESTADO, ESTADO, SIGLA) VALUES (4, 'AMAZONAS', 'AM');
INSERT INTO ESTADO (IDESTADO, ESTADO, SIGLA) VALUES (5, 'BAHIA', 'BA');
INSERT INTO ESTADO (IDESTADO, ESTADO, SIGLA) VALUES (6, 'CEARA', 'CE');
INSERT INTO ESTADO (IDESTADO, ESTADO, SIGLA) VALUES (7, 'DISTRITO FEDERAL', 'DF');
INSERT INTO ESTADO (IDESTADO, ESTADO, SIGLA) VALUES (8, 'ESPIRITO SANTO', 'ES');
INSERT INTO ESTADO (IDESTADO, ESTADO, SIGLA) VALUES (9, 'GOIAS', 'GO');
INSERT INTO ESTADO (IDESTADO, ESTADO, SIGLA) VALUES (10, 'MARANHAO', 'MA');
INSERT INTO ESTADO (IDESTADO, ESTADO, SIGLA) VALUES (11, 'MATO GROSSO', 'MT');
INSERT INTO ESTADO (IDESTADO, ESTADO, SIGLA) VALUES (12, 'MATO GROSSO DO SUL', 'MS');
INSERT INTO ESTADO (IDESTADO, ESTADO, SIGLA) VALUES (13, 'MINAS GERAIS', 'MG');
INSERT INTO ESTADO (IDESTADO, ESTADO, SIGLA) VALUES (14, 'PARA', 'PA');
INSERT INTO ESTADO (IDESTADO, ESTADO, SIGLA) VALUES (15, 'PARAIBA', 'PB');
INSERT INTO ESTADO (IDESTADO, ESTADO, SIGLA) VALUES (16, 'PARANA', 'PR');
INSERT INTO ESTADO (IDESTADO, ESTADO, SIGLA) VALUES (17, 'PERNAMBUCO', 'PE');
INSERT INTO ESTADO (IDESTADO, ESTADO, SIGLA) VALUES (18, 'PIAUI', 'PI');
INSERT INTO ESTADO (IDESTADO, ESTADO, SIGLA) VALUES (19, 'RIO DE JANEIRO', 'RJ');
INSERT INTO ESTADO (IDESTADO, ESTADO, SIGLA) VALUES
(20, 'RIO GRANDE DO NORTE', 'RN');
INSERT INTO ESTADO (IDESTADO, ESTADO, SIGLA) VALUES (21, 'RIO GRANDE DO SUL', 'RS');
INSERT INTO ESTADO (IDESTADO, ESTADO, SIGLA) VALUES (22, 'RONDONIA', 'RO');
INSERT INTO ESTADO (IDESTADO, ESTADO, SIGLA) VALUES (23, 'RORAIMA', 'RR');
INSERT INTO ESTADO (IDESTADO, ESTADO, SIGLA) VALUES (24, 'SANTA CATARINA', 'SC');
INSERT INTO ESTADO (IDESTADO, ESTADO, SIGLA) VALUES (25, 'SAO PAULO', 'SP');
INSERT INTO ESTADO (IDESTADO, ESTADO, SIGLA) VALUES (26, 'SERGIPE', 'SE');
INSERT INTO ESTADO (IDESTADO, ESTADO, SIGLA) VALUES (27, 'TOCANTINS', 'TO');

/*ENDERECOS*/
INSERT INTO ENDERECO (IDENDERECO, LOGRADOURO, BAIRRO, CIDADE, CEP, IDESTADO)
      VALUES (1, 'RUA 1', 'BAIRRO 1', 'RIO DE JANEIRO', '11111111', 25);
INSERT INTO ENDERECO (IDENDERECO, LOGRADOURO, BAIRRO, CIDADE, CEP, IDESTADO)
      VALUES (2, 'RUA 2', 'BAIRRO 2', 'SAO PAULO', '22222222', 13);
INSERT INTO ENDERECO (IDENDERECO, LOGRADOURO, BAIRRO, CIDADE, CEP, IDESTADO)
      VALUES (3, 'RUA 3', 'BAIRRO 3', 'FRANCA', '11111111', 12);

/*CLIENTES*/
INSERT INTO CLIENTE (IDCLIENTE, NOME, TIPO, NUMERO, IDENDERECO)
      VALUES (1, 'DEVMEDIA ', 'J', 1234, 2);
INSERT INTO CLIENTE (IDCLIENTE, NOME, TIPO, NUMERO, IDENDERECO)
      VALUES (2, 'FOX', 'J', 5678, 3);
INSERT INTO CLIENTE (IDCLIENTE, NOME, TIPO, NUMERO, IDENDERECO)
      VALUES (3, 'DANIEL SOBRINHO LAPORTE', 'F', 1212, 1);
```

parâmetro 'D', na qual realizaria a exclusão dos registros da tabela de estado que não estão em uso, ou mesmo a criação das chaves estrangeiras por meio de outro caractere' qualquer. Agora que o banco de dados já está criado, executando o código de mapeamento criado na **Listagem 2** é possível realizar o mapeamento, conforme ilustra a **Figura 2.**

	IDTABELA	NOME_TABELA	VISIVEL
▶	1	CLIENTE	S
	2	ENDERECO	S
	3	ESTADO	S
	4	PESSOA_FISICA	S
	5	PESSOA_JURIDICA	S

Figura 2. Mapeamento realizado por meio das tabelas de sistema

Utilizando estes recursos dentro do Delphi

Como se sabe, na versão 2010, a Embarcadero disponibilizou no DBX um driver de conexão nativo para o banco de dados Firebird, tornando o DBX Framework ainda mais seletivo. Embora os usuários a partir de então tenham acesso a um Driver nativo, nem todos os recursos são suportados. Como é o caso do Execute Block. Porém isto pode ser contornado com outro componente de acesso a banco de dados, como as suítes FIBPlus ou IBObjects, ou mesmo um outro driver de conexão nativa como o driver da DevArt por exemplo. A **Figura 3** demonstra o que ocorre em uma tentativa de criação/execução de um Execute Block por meio do DBX com um SQLDataSet.

Nota-se que as variáveis internas (V_IDCLIENTE e V_TIPO) são mapeadas como parâmetros pelo DBX, o que em si já impossibilita o preenchimento destes parâmetros já que são alimentados internamente pelo próprio Execute Block. Se ainda assim necessitar utilizar o Execute Block com o DBX, ainda há uma alternativa. Utilizar a execução direta pelo SQLConnection como a seguir:

```
SQLConnection1.ExecuteDirect(SQLDataSet.CommandText);
```

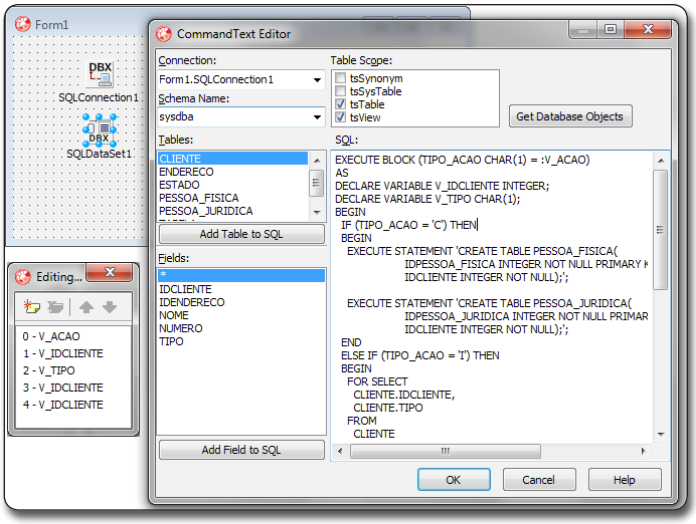


Figura 3. Execute Block em um SQLDataSet

Neste caso o conteúdo inserido no SQLDataSet (desde que não haja parâmetros) pode ser executado normalmente. Caso haja parâmetros o procedimento pode ser montado dinamicamente por meio de uma String e inserido da mesma maneira.

Listagem 4. Alteração do Execute Block

```
EXECUTE BLOCK (TIPO_ACAO CHAR(1) = :V_ACAO)
AS
DECLARE VARIABLE V_IDCLIENTE INTEGER;
DECLARE VARIABLE V_TIPO CHAR(1);
BEGIN
 IF (TIPO_ACAO = 'C') THEN
 BEGIN
  EXECUTE STATEMENT 'CREATE TABLE PESSOA_FISICA(
            IDPESSOA_FISICA INTEGER NOT NULL PRIMARY KEY ,
            IDCLIENTE INTEGER NOT NULL);';

  EXECUTE STATEMENT 'CREATE TABLE PESSOA_JURIDICA(
            IDPESSOA_JURIDICA INTEGER NOT NULL PRIMARY KEY ,
            IDCLIENTE INTEGER NOT NULL);';
 END
 ELSE IF (TIPO_ACAO = 'I') THEN
 BEGIN
  FOR SELECT
   CLIENTE.IDCLIENTE,
   CLIENTE.TIPO
  FROM
   CLIENTE
  INTO
   :V_IDCLIENTE,
   :V_TIPO
  DO
  BEGIN
   IF (V_TIPO = 'F') THEN
    EXECUTE STATEMENT '
     INSERT INTO PESSOA_FISICA (
      IDPESSOA_FISICA,
      IDCLIENTE)
     VALUES (
      (SELECT COALESCE(MAX(IDPESSOA_FISICA), 0) + 1
       FROM PESSOA_FISICA), ' ||
      :V_IDCLIENTE || ');';
   ELSE
    EXECUTE STATEMENT '
     INSERT INTO PESSOA_JURIDICA (
      IDPESSOA_JURIDICA,
      IDCLIENTE)
     VALUES (
      (SELECT COALESCE(MAX(IDPESSOA_JURIDICA), 0) + 1
       FROM PESSOA_JURIDICA), ' ||
      :V_IDCLIENTE || ');';
  END
 END
END
```

Conclusão

Durante o artigo foram exibidos modelos de Procedures e Execute Blocks como uma forma de comparação e é claro, proporcionando uma facilidade de memorização e compreensão de como utilizar o recurso. Foram abordados casos de utilização do recurso abordado unido ao conceito de tabelas de sistema, na qual exige um conhecimento específico maior, porém, pode se tornar muito mais produtivo se utilizado de forma dinâmica por meio do Execute Block. Além disto, foi possível avaliar uma situação de mudanças em um banco de dados, podendo visualizar como proceder com este tipo de caso.

Concluindo, o Execute Block é um recurso que somado a criatividade, pode proporcionar grande evolução de código, facilidade de manutenção e atualização de bancos dados independente de estrutura ou dados. Contudo, deve ser utilizado com cautela, pois, apesar de ser um procedimento dinâmico, possui velocidade inferior (mesmo que pouco perceptível) em relação a um objeto pré-compilado no banco de dados, o que pode acarretar em futuras dores de cabeça dependendo dos casos. Um grande abraço e até a próxima!

Daniel Sobrinho Laporte
daniel.laporte@gmail.com / www.twitter.com/daniel_laporte
Colunista e editor técnico da Revista ClubeDelphi. Graduado em Sistemas de informação e pós-graduado em Administração de Bancos de Dados. É analista de sistemas e desenvolvedor Delphi desde 2005 desenvolvendo aplicações cliente/servidor e multicamadas.

Site oficial do Firebird
http://www.firebirdsql.org/

Site oficial dos componentes IBObjects
http://www.ibobjects.com/

Site oficial dos componentes FIBPlus
http://www.devrace.com/en/fibplus/

Site oficial da DevArt
http://www.devart.com/

Dê seu feedback sobre esta edição!

A Clubedelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre este artigo, através do link:

www.devmedia.com.br/clubedelphi/feedback

Design Patterns
Utilizando os padrões Decorator

Resumo DevMan

De que trata o artigo:

O objetivo deste artigo é discutir cenários de implementação em que poderão estar sendo empregados os patterns Decorator e Singleton.

Em que situação o tema é útil:

Certas situações dentro do desenvolvimento de software podem ser contornadas de maneira mais fácil por meio do uso dos design patterns. Um padrão corresponde a uma técnica bem sucedida e direcionada a um problema específico, possuindo ainda uma eficiência comprovada por sucessivas implementações do mesmo em numerosos projetos.

Design Patterns - Utilizando os padrões Decorator e Singleton:

Determinadas situações podem se repetir com certa regularidade dentro do desenvolvimento de software. Em tais casos, não será raro que outros desenvolvedores tenham passado pelas mesmas dificuldades, sendo que aspectos como estes po dem ter levado à elaboração de padrões visando solucionar destes problemas específicos.

O padrão Decorator procura ser uma alternativa para a inclusão (ou alteração) de comportamentos de um objeto de maneira dinâmica, sem que isto force a mudanças na estrutura original considerada. Já o pattern Singleton fornece diretrizes para a existência de uma única instância de uma classe durante todo o ciclo de vida de uma aplicação.

Este artigo continua a abordagem iniciada em edições anteriores a respeito do uso de design patterns na área de software. Para isto serão discutidos aqui cenários em que os padrões Decorator e Singleton podem ser empregados, assim como características e princípios próprios de cada um destes patterns.

Design patterns (ou em português "padrões de projetos") são técnicas destinadas à resolução de problemas específicos, os quais podem acontecer com alguma frequência durante a codificação de sistemas. O uso de padrões em projetos de software objetiva:

- Um código menos extenso e mais fácil de ser compreendido;
- Maior flexibilidade de uma aplicação frente à necessidade de mudanças;
- Uma clara separação de responsabilidades entre as diferentes classes que formam uma aplicação, evitando com isto construções de código complexas;
- O reuso de estruturas tanto dentro da aplicação que serviu de base para o surgimento das mesmas, como também em outros projetos que dependam de funcionalidades similares.

Padrões de projetos não têm sua utilização restrita a uma tecnologia específica, mesmo porque procuram ser uma solução genérica para um determinado tipo de questão. Graças a este tipo de preocupação, um pattern pode vir a ser aplicado nas mais diferentes plataformas de desenvolvimento (Delphi, C#, VB.NET, Java, C++, etc.).

Podem ser listados como benefícios decorrentes do uso de padrões:

- Uma maior padronização do código-fonte, uma vez que as implementações

e Singleton

Renato José Groffe

são realizadas levando em conta boas práticas de Orientação a Objetos;
• Aplicações de software formadas por componentes mais coesos;
• A possibilidade de reuso de algumas estruturas baseadas em design patterns;
• Por contarem normalmente com extensa documentação, padrões de software podem ser assimilados mais rapidamente por membros de uma equipe de desenvolvimento.

É uma prática comum que muitos patterns sejam representados graficamente por meio da notação UML. Diagramas de classes são o tipo de recurso mais empregado nestes casos.

Nota do DevMan

UML (sigla em inglês para "Unified Modeling Language") é uma linguagem voltada à modelagem de componentes de software e que se baseia em conceitos de orientação a objetos. Trata-se de um notação que disponibiliza uma série de diagramas, sendo o de classes um dos mais conhecidos (com este último sendo utilizado para a demonstração dos diversos patterns descritos neste artigo).

Adicionando comportamentos dinamicamente com o padrão Decorator

O padrão estrutural Decorator (**Figura 1**) busca permitir a adição de um comportamento adicional a um objeto de maneira dinâmica, ou seja, em tempo de execução. Um objeto baseado neste pattern adiciona assim um tratamento específico a uma instância de classe, dispensando, no entanto a necessidade de alterar o código existente. Uma forma comum de se implementar esta prática é por meio do uso de subclasses que implementam um tipo básico.

As seguintes diretrizes servem de base para a implementação deste padrão:
• Uma classe abstrata (ou dependendo da situação uma interface), na qual existirá uma série de comportamentos esperados (Componente);
• Classes concretas baseadas no tipo básico (classe ComponenteConcreto no exemplo da figura);
• Um "decorador", ou seja, um tipo (classe abstrata Decorator) que servirá de base para estender os comportamentos da classe abstrata básica;
• Classes concretas que derivam do "decorador" (DecoratorConcretoA e DecoratorConcretoB).

A utilização de um Decorator acontece de forma transparente aos consumidores de um tipo básico, já que estes desconhecem implementações específicas da classe-base considerada. Além de ser comumente empregado na inclusão de um novo comportamento dinâmico, um Decorator pode ainda redefinir uma operação dentro de sua implementação (fazendo uso então de sobrecarga do método considerado, por exemplo).

O padrão Decorator deve ser utilizado com critério, já que a aplicação do mesmo pode resultar em um grande número de subclasses, contribuindo para uma maior complexidade das estruturas presentes em um projeto.

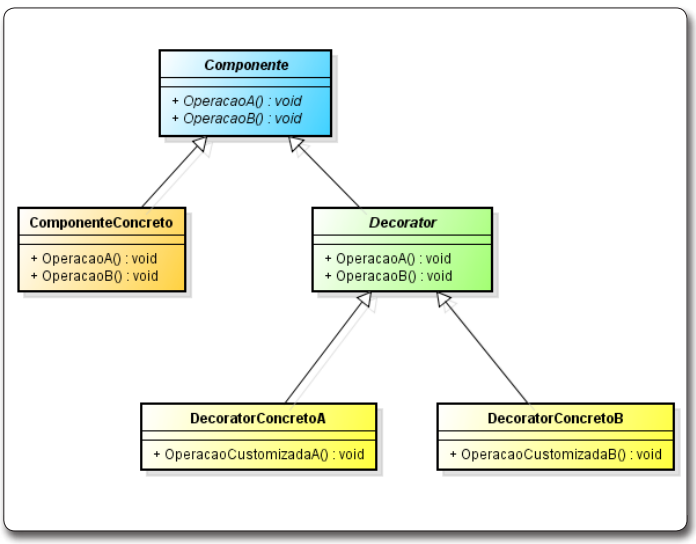


Figura 1. Representação esquemática do pattern Decorator

Nota

O diagrama de classe apresentado nesta seção foi criado a partir da aplicação Astah UML. Esta ferramenta disponibiliza tanto versões gratuitas quanto proprietárias, sendo que na seção de Links encontra-se o endereço a partir do qual o download da mesma poderá ser realizado.

Nota do DevMan

O pattern Decorator pode ser visto uma aplicação do Princípio Aberto-Fechado (ou OCP, sigla do inglês "Open/closed principle"). Este conceito enfatiza a necessidade de construírem classes que possam ser estendidas sem grandes dificuldades, de forma a evitar que a definição de um novo comportamento no futuro modifique estruturas pré-existentes.

Garantindo a existência de uma única instância com o padrão Singleton

Dentre todos os padrões GoF existentes, o pattern de criação Singleton talvez seja a solução mais simples de se implementar. A finalidade deste padrão é assegurar a existência de uma única instância de uma classe ao longo da execução de uma aplicação. Esta característica tem como implicação direta o fato de que diferentes de um sistema, ao utilizarem um tipo que implementa este padrão, apontarão sempre para uma mesma referência de objeto.

O uso do padrão Singleton pode ser incentivado por fatores como a busca por uma melhor performance (impedindo assim a criação de diversas instâncias que consumiriam recursos de forma desmasiada) ou, ainda, pela necessidade de se limitar o acesso à única instância que deverá existir para um determinado tipo (através de um ponto centralizado para obtenção da mesma).

As seguintes diretrizes deverão ser consideradas para a implementação de uma classe baseada no padrão Singleton:

- Deve-se proibir o acesso ao construtor do tipo, impedindo assim que outros elementos criem diretamente referências de objetos baseados no mesmo. Logo, o construtor em questão estará marcado como privado;
- Um método público e de classe será implementado, funcionado como ponto de acesso centralizado à única referência existente dentro de uma aplicação. Isto envolverá também o uso de uma variável também de classe (porém privada), a qual representa a instância única da classe Singleton.

Nota do DevMan

Variáveis e métodos de classe são estruturas que não dependem da existência de uma instância de um tipo. Tais construções estarão disponíveis para acesso enquanto uma aplicação estiver em execução na memória. Elementos com essas características também são conhecidos como membros estáticos de uma classe.

Exemplos de uso dos patterns Decorator e Singleton

A implementação das aplicações demonstradas neste artigo foi realizada através da utilização do Embarcadero Delphi XE2, sendo que por meio das mesmas serão apresentadas formas de uso dos patterns Decorator e Singleton.

Utilizando o pattern Decorator

A construção de uma aplicação empregando o padrão Decorator inicia-se com a criação de um projeto do tipo "VCL Forms Application" chamado TesteDecorator, sendo que o mesmo envolverá a geração de um arquivo de contatos empresariais nos formatos .txt e .csv (formato delimitado por ponto-e-vírgula). Inicialmente será desenvolvida uma classe que permite a exportação de informações de contato para um arquivo .txt. Implementa-se então um segundo tipo que estende essa estrutura inicial, de maneira que o mesmo possibilite a geração de um arquivo .csv com os dados existentes.

Criação das classes que empregam o padrão Decorator

Na **Listagem 1** está o código-fonte correspondente à classe abstrata TArquivoContatos. Este tipo define os comportamentos possíveis para uma estrutura que será responsável pela geração de arquivos de contatos num formato específico.

O único método concreto de TArquivoContatos é GerarArquivo, sendo que essa operação emprega uma lista de strings (objeto do tipo TStringList) para a montagem de um novo arquivo. Visando cumprir o objetivo para o qual foi definido, GerarArquivo faz uso dos métodos abstratos IniciarProcessamentoArquivo, ExistemContatos e ProcurarProximoContato.

Listagem 1. Classe abstrata TArquivoContatos

```
type
 TArquivoContatos = class
 public
   procedure IniciarProcessamentoArquivo; virtual; abstract;
   function ExistemContatos: boolean; virtual; abstract;
   function ObterLinhaArquivoFormatada: string; virtual; abstract;
   function ObterRazaoSocialAtual: string; virtual; abstract;
   function ObterNomeContatoAtual: string; virtual; abstract;
   function ObterTelefoneAtual: string; virtual; abstract;
   function ObterEmailAtual: string; virtual; abstract;
   procedure ProcurarProximoContato; virtual; abstract;
   function ObterNomeArquivoExportacao: string; virtual; abstract;
   procedure GerarArquivo;
 end;

implementation

uses System.Classes;

{ TArquivoContatos }

procedure TArquivoContatos.GerarArquivo;
var
   stlLinhasArquivo: TStringList;
begin
 stlLinhasArquivo:= TStringList.Create;

 IniciarProcessamentoArquivo;
 while (ExistemContatos) do
 begin
   stlLinhasArquivo.Add(ObterLinhaArquivoFormatada);
   ProcurarProximoContato;
 end;

 stlLinhasArquivo.SaveToFile(ObterNomeArquivoExportacao);
end;
```

A seguir estão listadas todas as operações abstratas declaradas em TArquivoContatos, bem como as respectivas atribuições de cada método:

- IniciarProcessamentoArquivo: executa instruções para se iniciar o processo de geração de um novo arquivo;
- ExistemContatos: retorna um valor booleano que indica se existem outros contatos a serem gravados num arquivo;
- ObterLinhaArquivoFormatada: devolve uma string contendo um registro de contato num formato específico;
- ObterRazaoSocialAtual: empresa do contato atual;
- ObterNomeContatoAtual: nome do contato atual;

• ObterTelefoneAtual: telefone do contato atual;
• ObterEmailAtual: e-mail do contato atual;
• ProcurarProximoContato; busca o próximo contato dentro de uma sequência de informações deste tipo;
• ObterNomeArquivoExportacao: nome do arquivo de exportação que será gerado a partir das informações de contatos.

A **Listagem 2** exibe o código que implementa a classe TArquivoContatosTXT. Essa estrutura será responsável por gerar um arquivo no formato .txt, tomando como base para isto um objeto do tipo TClientDataSet com informações relativas a contatos. Tal instância de TClientDataSet será obtida através do construtor de TArquivoContatosTXT, sendo armazenada na variável privada cdsContatos para uso posterior.

Importante destacar ainda que TArquivoContatosTXT implementa a classe abstrata TArquivoContatos, com as operações correspondentes dependendo de instruções que empregam o objeto cdsContatos. Dentro da função ObterLinhaArquivoFormatada ocorrerá a formatação das linhas que constituirão um arquivo-texto, com cada uma das mesmas tendo seus campos separados pelo caracter "|" (pipe).

Já a **Listagem 3** apresenta o código-fonte do tipo TArquivoContatosCSV. Este último funcionará como um Decorator para instâncias de TArquivoContatosTXT (ou qualquer outra estrutura que venha a implementar TArquivoContatos), com isso acontecendo através de um construtor que recebe uma instância de TArquivoContatos: esta referência será então armazenada na variável privada tipoArquivoOriginal.

Boa parte dos métodos definidos em TArquivoContatosCSV e que são derivados de TArquivoContatos estão delegando chamadas às operações correspondentes em TArquivoContatosTXT (por meio da variável tipoArquivoOriginal). A única exceção a isto são as funções ObterLinhaArquivoFormatada e ObterNomeArquivoExportacao. No caso específico de ObterLinhaArquivoFormatada, utiliza-se como separador de colunas o caracter ";" (ponto-e-vírgula), sendo esta uma característica específica de arquivos .csv.

Implementação da função que retorna o ClientDataSet de Contatos

Na **Listagem 4** consta o código que define a função ObterClient DataSetContatos. Este método emprega um objeto TClientDataSet

Listagem 2. Classe TArquivoContatosTXT

```
uses ArquivoContatos, Datasnap.DBClient;

type
 TArquivoContatosTXT = class(TArquivoContatos)
 private
  cdsContatos: TClientDataSet;
 public
  constructor Create(cdsContatosEmpresariais: TClientDataSet);
  procedure IniciarProcessamentoArquivo; override;
  function ExistemContatos: boolean; override;
  function ObterRazaoSocialAtual: string; override;
  function ObterNomeContatoAtual: string; override;
  function ObterTelefoneAtual: string; override;
  function ObterEmailAtual: string; override;
  function ObterLinhaArquivoFormatada: string; override;
  procedure ProcurarProximoContato; override;
  function ObterNomeArquivoExportacao: string; override;
 end;

implementation

{ TArquivoContatosTXT }

constructor TArquivoContatosTXT.Create(
 cdsContatosEmpresariais: TClientDataSet);
begin
 cdsContatos:= cdsContatosEmpresariais;
end;

procedure TArquivoContatosTXT.IniciarProcessamentoArquivo;
begin
 cdsContatos.First;
end;

function TArquivoContatosTXT.ExistemContatos: boolean;
begin
 Result:= (Not cdsContatos.Eof);
end;

function TArquivoContatosTXT.ObterRazaoSocialAtual: string;
begin
 Result:= cdsContatos.FieldByName('RAZAO_SOCIAL').AsString;
end;

function TArquivoContatosTXT.ObterNomeContatoAtual: string;
begin
 Result:= cdsContatos.FieldByName('NOME_CONTATO').AsString;
end;

function TArquivoContatosTXT.ObterTelefoneAtual: string;
begin
 Result:= cdsContatos.FieldByName('TELEFONE').AsString;
end;

function TArquivoContatosTXT.ObterEmailAtual: string;
begin
 Result:= cdsContatos.FieldByName('EMAIL').AsString;
end;

function TArquivoContatosTXT.ObterLinhaArquivoFormatada: string;
begin
 Result:= '|' + cdsContatos.FieldByName('RAZAO_SOCIAL').AsString + '|' +
  cdsContatos.FieldByName('NOME_CONTATO').AsString + '|' +
  cdsContatos.FieldByName('TELEFONE').AsString + '|' +
  cdsContatos.FieldByName('EMAIL').AsString + '|';
end;

procedure TArquivoContatosTXT.ProcurarProximoContato;
begin
 cdsContatos.Next;
end;

function TArquivoContatosTXT.ObterNomeArquivoExportacao: string;
begin
 Result:= 'contatos.txt';
end;
```

criado em memória, sendo que no mesmo estarão informações referentes a diversos contatos empresariais. A inclusão de dados de contatos no ClientDataSet acontece através do procedimento auxiliar IncluirContato.

Implementação da tela para testes do padrão Decorator

Uma vez que o processo de codificação das estruturas relativas à obtenção de dados de contatos e geração de arquivos tenha finalizado, deve-se prosseguir com a implementação da tela que será empregada nos testes destes elementos. O formulário principal da aplicação deve ser renomeado para FormTeste. Na **Listagem 5** é apresentado o código-fonte contendo a implementação do mesmo.

O método btnGerarArquivoClick é disparado quando da execução do evento OnClick do botão btnGerarArquivo. Esta operação gera inicialmente uma instância do tipo TArquivoContatosTXT (com o construtor deste recebendo um objeto ClientDataSet através da função ObterClientDataSetContatos), com esta referência sendo atribuída à variável de nome "arquivo". Caso tenha sido marcada a opção para a exportação dos contatos para o formato .txt, o objeto associado à variável "arquivo" será a base para a geração do arquivo correspondente.

Por outro lado, se for marcada a opção para a geração de arquivos no formato .csv, a instância vinculada à variável "arquivo" é repassada ao construtor de TArquivoContatosCSV. A nova referência (baseada em TArquivoContatosCSV) funcionará como um Decorator do objeto original, sobrepondo o comportamento de geração de dados em .txt pelo equivalente no padrão .csv.

Executando a aplicação para testes do padrão Decorator

Ao se executar o projeto TesteDecorator será exibida uma tela como a indicada na **Figura 2**, na qual deverá ser selecionada alguma das opções para a geração de arquivos de contatos (no formato .txt ou .csv).

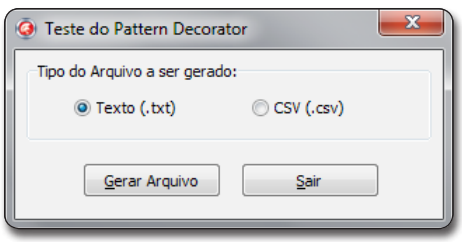


Figura 2. Tela inicial que é exibida ao se executar o projeto TesteDecorator

Listagem 3. Classe TArquivoContatosCSV

```
uses ArquivoContatos;

type
 TArquivoContatosCSV = class(TArquivoContatos)
 private
  tipoArquivoOriginal: TArquivoContatos;
 public
  constructor Create(arquivoBase: TArquivoContatos);
  procedure IniciarProcessamentoArquivo; override;
  function ExistemContatos: boolean; override;
  function ObterRazaoSocialAtual: string; override;
  function ObterNomeContatoAtual: string; override;
  function ObterTelefoneAtual: string; override;
  function ObterEmailAtual: string; override;
  function ObterLinhaArquivoFormatada: string; override;
  procedure ProcurarProximoContato; override;
  function ObterNomeArquivoExportacao: string; override;
 end;

implementation

{ TArquivoContatosCSV }

constructor TArquivoContatosCSV.Create(arquivoBase: TArquivoContatos);
begin
 tipoArquivoOriginal:= arquivoBase;
end;

procedure TArquivoContatosCSV.IniciarProcessamentoArquivo;
begin
 tipoArquivoOriginal.IniciarProcessamentoArquivo;
end;

function TArquivoContatosCSV.ExistemContatos: boolean;
begin
 Result:= tipoArquivoOriginal.ExistemContatos;
end;

function TArquivoContatosCSV.ObterRazaoSocialAtual: string;
begin
 Result:= tipoArquivoOriginal.ObterRazaoSocialAtual;
end;

function TArquivoContatosCSV.ObterNomeContatoAtual: string;
begin
 Result:= tipoArquivoOriginal.ObterNomeContatoAtual;
end;

function TArquivoContatosCSV.ObterTelefoneAtual: string;
begin
 Result:= tipoArquivoOriginal.ObterTelefoneAtual;
end;

function TArquivoContatosCSV.ObterEmailAtual: string;
begin
 Result:= tipoArquivoOriginal.ObterEmailAtual;
end;

function TArquivoContatosCSV.ObterLinhaArquivoFormatada: string;
begin
 Result:= tipoArquivoOriginal.ObterRazaoSocialAtual + ';' +
  tipoArquivoOriginal.ObterNomeContatoAtual + ';' +
  tipoArquivoOriginal.ObterTelefoneAtual + ';' +
  tipoArquivoOriginal.ObterEmailAtual;
end;

procedure TArquivoContatosCSV.ProcurarProximoContato;
begin
 tipoArquivoOriginal.ProcurarProximoContato;
end;

function TArquivoContatosCSV.ObterNomeArquivoExportacao: string;
begin
 Result:='contatos.csv';
end;
```

Listagem 4. Implementação da função ObterClientDataSetContatos

```
procedure IncluirContato(cds: TClientDataSet;
 razaoSocial, nomeContato, telefone, email: string);
begin
 cds.Append;
 cds.FieldByName('RAZAO_SOCIAL').AsString:= razaoSocial;
 cds.FieldByName('NOME_CONTATO').AsString:= nomeContato;
 cds.FieldByName('TELEFONE').AsString:= telefone;
 cds.FieldByName('EMAIL').AsString:= email;
 cds.Post;
end;
function ObterClientDataSetContatos: TClientDataSet;
var
  cds: TClientDataSet;
begin
 cds:= TClientDataSet.Create(nil);
 cds.FieldDefs.Add('RAZAO_SOCIAL', ftString, 50);
 cds.FieldDefs.Add('NOME_CONTATO', ftString, 50);
 cds.FieldDefs.Add('TELEFONE', ftString, 13);
 cds.FieldDefs.Add('EMAIL', ftString, 50);
 cds.CreateDataSet;
 IncluirContato(cds, 'EMPRESAS ACME LTDA',
  'JOAO DA SILVA', '(11)2529-2784', 'joao.silva@acme.com.br');
 IncluirContato(cds, 'MÓVEIS PAULISTA S.A.',
  'JOSÉ MARTINS', '(11)2345-8778', 'jose.martins@mpaulista.com.br');
 IncluirContato(cds, 'RST ENGENHARIA LTDA.',
  'JOAQUIM DE OLIVEIRA', '(19)4758-5760', 'joaquim.oliveira@rst.com.br');
 IncluirContato(cds, 'TRANSPORTES OURO FINO ME',
  'MARIA BATISTA', '(31)4512-7814', 'maria.batista@ourofino.com.br');
 IncluirContato(cds, 'VIAGENS AZUIS LTDA.',
  'PEDRO LEME', '(21)3783-5797', 'joao.silva@viagensazuis.com.br');
 Result:= cds;
end;
```

Listagem 5. Implementação do formulário TFormTeste

```
procedure TFormTeste.btnGerarArquivoClick(Sender: TObject);
var
  arquivo: TArquivoContatos;
begin
 arquivo:= TArquivoContatosTXT.Create(ObterClientDataSetContatos);
 if (rdbArquivoCSV.Checked) then
  arquivo:= TArquivoContatosCSV.Create(arquivo);
 arquivo.IniciarProcessamentoArquivo;
 arquivo.GerarArquivo;
 ShowMessage('Arquivo ' + arquivo.ObterNomeArquivoExportacao + ' ' +
  'gerado com sucesso!');
end;
```

Caso esteja selecionada a opção "Texto (.txt)", o usuário irá se deparar com a janela apresentada na **Figura 3**. Já a **Figura 4** apresenta o arquivo .txt que foi gerado e, neste caso, carregado a partir do Bloco de Notas do Windows.

Se optar pela criação de um arquivo .csv será apresentada ao usuário a mensagem que consta na **Figura 5**. Já a **Figura 6** exibe o arquivo .csv já montado e, neste caso, aberto dentro do Excel.

Utilizando o pattern Singleton

Para se testar o uso do Singleton deverá ser criado um projeto do tipo "VCL Forms Application" chamado TesteSingleton. A partir do executável resultante será possível simular a obtenção de um empréstimo junto a um banco/financeira, informando para isto o valor do mesmo e a taxa de juros mensais; com isto será possível obter o total equivalente em doze meses aplicando juros compostos.

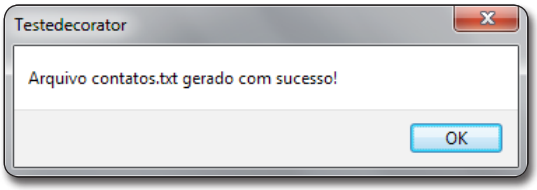


Figura 3. Aviso de que o arquivo .txt foi gerado com sucesso

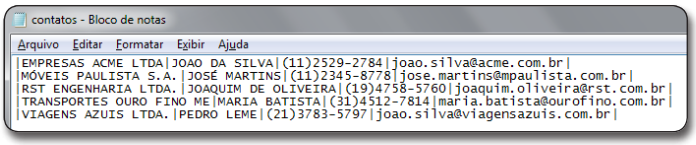


Figura 4. Arquivo contatos.txt

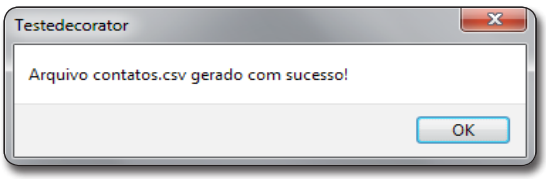


Figura 5. Aviso de que o arquivo .csv foi gerado com sucesso

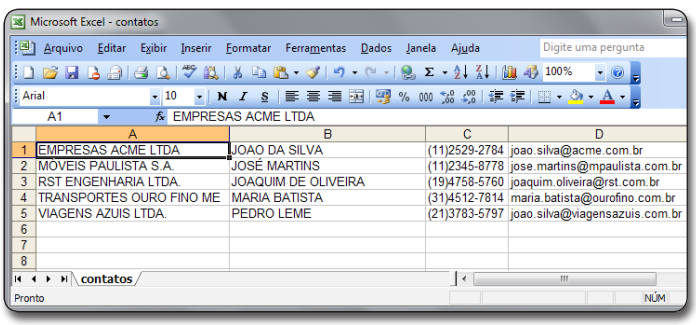


Figura 6. Arquivo contatos.csv

Um mecanismo de log baseado no padrão Singleton será implementado, com este recurso servindo de base para o registro de erros ou, ainda, ações que foram executadas com sucesso (inicialização/término da aplicação, simulações que não apresentaram problemas, etc.).

Criação das classe que emprega o padrão Singleton

A **Listagem 6** apresenta o código-fonte da classe TLog. Essa estrutura corresponde ao mecanismo de log para registro de ações (de erro ou não) desencadeadas dentro do executável de simulação, sendo que tais informações serão reflexo de interações de um usuário com a aplicação TesteSingleton. Rotinas de log representam um bom exemplo de um termo computacional conhecido como "crosscutting concerns".

A respeito da implementação do tipo TLog, é possível observar:

- A existência de um construtor privado (Create), sendo que impedir o acesso a tal estrutura de uma classe é uma das premissas do padrão Singleton;
- A variável privada stlRegistrosLog. Este objeto do tipo TString List será utilizado para o armazenamento das diferentes mensagens de log e, conseqüentemente, a gravação das mesmas em um

Listagem 6. Classe TLog

```
type
 TLog = class
 private
  class var log: TLog;
  stlRegistrosLog: TStringList;
  constructor Create;
  procedure RegistrarMensagemLog(mensagem: string);
 public
  class function GetInstance: TLog;
  procedure RegistrarEvento(mensagemEvento: string);
  procedure RegistrarErro(mensagemErro: string);
 end;

implementation

uses System.SysUtils;

{ TLog }

constructor TLog.Create;
begin
 stlRegistrosLog:= TStringList.Create;
 if FileExists('log.csv') then
  stlRegistrosLog.LoadFromFile('log.csv');
end;

procedure TLog.RegistrarMensagemLog(mensagem: string);
begin
 stlRegistrosLog.Add(mensagem);
 stlRegistrosLog.SaveToFile('log.csv');
end;

class function TLog.GetInstance: TLog;
begin
 if (log = nil) then
  log:= TLog.Create;
 Result:= log;
end;

procedure TLog.RegistrarEvento(mensagemEvento: string);
begin
 RegistrarMensagemLog(DateTimeToStr(Now) + ';' +
  'Informação;' + mensagemEvento)
end;

procedure TLog.RegistrarErro(mensagemErro: string);
begin
 RegistrarMensagemLog(DateTimeToStr(Now) + ';' +
  'Erro;' + mensagemErro)
end;
```

arquivo no formato .csv de nome log.csv (e que estará no mesmo diretório em que se encontrar o executável da aplicação);

• O método privado RegistrarMensagemLog. Esta operação será responsável por adicionar uma nova mensagem à lista de registros de log (variável stlRegistrosLog);

• RegistrarEvento e RegistrarErro são operações similares, com o primeiro destes métodos sendo utilizado para registrar eventos como o encerramento da aplicação num dado instante, ao passo que RegistrarErro servirá para a gravação de informações sobre erros que tenham ocorrido na aplicação. Estes dois métodos usam a operação RegistrarMensagemLog para a gravação de mensagens no arquivo de log;

• A variável de classe log armazenará em memória a única instância possível para a classe TLog. Importante que log foi declarada como privada, de maneira que se impeça o acesso, ou mesmo, a atribuição de um valor nulo à mesma a partir de outra estrutura;

• GetInstance é um método de classe, sendo que o mesmo representa o ponto de acesso centralizado à instância única de TLog (e por meio da qual a variável log será inicializada ao se acessar pela primeira vez esta operação).

Nota do DevMan

Crosscutting concerns (em português "interesses transversais") é um conceito que se refere a funções genéricas, sem uma ligação direta com aspectos de negócio e cujo uso encontra-se disseminado ao longo de uma aplicação. Constituem exemplos notórios disto mecanismos como autenticação / autorização de usuários, tratamento de exceções, logging, dentre outras funcionalidades. Separar recursos deste tipo em classes contribui não apenas para evitar a duplicação de código por todo um sistema, como também permite o reuso das estruturas criadas em uma série de projetos.

Implementação da função para cálculo de Juros Compostos

Na **Listagem 7** está o código-fonte contendo a implementação da função CalcularVlEmprestimo12Meses. Este método será responsável por calcular, a partir do fornecimento do valor de um empréstimo e taxa percentual de juros mensais, o montante a ser pago ao final de um período de doze meses (usando para isto juros compostos). Nota-se no código-fonte, ainda, o uso da função Power, a qual é empregada em cálculos que envolvam operações de exponenciação.

Listagem 7. Implementação da função CalcularVlEmprestimo12Meses

```
uses System.Math;

function CalcularVlEmprestimo12Meses(rVlEmprestimo: Real;
 rVlPercMensalJuros: Real): Real;
begin

 Result:= rVlEmprestimo * Power(1 + (rVlPercMensalJuros / 100), 12);
end;
```

Implementação da tela para testes do padrão Singleton

Assim que se concluírem as atividades de codificação das estruturas envolvidas no uso do pattern Singleton, chega o momento de se construir a tela através da qual serão efetuados os testes. O formulário principal da aplicação terá sua propriedade Name configurada com o valor "FormTeste". A **Listagem 8** apresenta a implementação desta tela.

É partir do evento btnCalcularClick que a função CalcularVlEmprestimo12Meses é acionada, visando com isto obter o valor total a ser pago para um empréstimo dentro de 12 meses.

Listagem 8. Implementação do formulário TFormTeste

```
procedure TFormTeste.FormCreate(Sender: TObject);
var
  log: TLog;
begin
 log:= TLog.GetInstance();
 log.RegistrarEvento('Iniciada a aplicação de simulação de empréstimos.');
end;

procedure TFormTeste.btnCalcularClick(Sender: TObject);
var
  log: TLog;
  rVlEmprestimo: Real;
  rVlPercMensalJuros: Real;
  rVlTotalFimPeriodo: Real;
begin

 log:= TLog.GetInstance();
 try
  rVlEmprestimo:= StrToFloat(txtVlEmprestimo.Text);
  if (rVlEmprestimo <= 0) then
  begin
   ShowMessage('O Valor do Empréstimo deve ser maior do que zero!');
   log.RegistrarErro('Valor do Empréstimo inválido.');
   Exit;
  end;

  rVlPercMensalJuros:= StrToFloat(txtPercMensalJuros.Text);
  if (rVlPercMensalJuros <= 0) Or
    (rVlPercMensalJuros > 100) then
  begin
   ShowMessage('O Percentual Mensal de Juros deve ser maior ' +
    'do que zero e mnor do que 100,00%!');
   log.RegistrarErro('Percentual Mensal de Juros inválido.');
   Exit;
  end;

  rVlTotalFimPeriodo:= CalcularVlEmprestimo12Meses(
   rVlEmprestimo, rVlPercMensalJuros);
  ShowMessage('Valor Total ao Final de um Período de 12 Meses: ' +
   FormatFloat('0.00', rVlTotalFimPeriodo));
  log.RegistrarEvento('Cálculo da simulação de empréstimo ' +
   'efetuado com sucesso.');
 except
  ShowMessage('Ocorreu um erro durante a simulação. Verifique ' +
   'se os valores informados são válidos e repita o processo ' +
   'novamente.');
  log.RegistrarErro('Erro durante a simulação de empréstimo.');
 end;
end;

procedure TFormTeste.FormClose(Sender: TObject; var Action: TCloseAction);
var
  log: TLog;
begin
 log:= TLog.GetInstance;
 log.RegistrarEvento('Encerrada a aplicação de simulação de empréstimos.');
end;
```


CURSOS ONLINE

A Revista .net Magazine oferece aos seus assinantes uma série de Cursos Online de alto padrão de qualidade.

.net magazine

CONHEÇA OS CURSOS MAIS RECENTES:

- Curso .net Entity Framework - Loja Virtual
- Novidades do Visual Studio 2010
- Componentes DevExpress
- Crie um chat utilizando Asp.Net 3.5 e Linq
- Novidades do C# (csharp) 3.0
- POO com C#
- Microsoft Visual Studio Tools for Office (VSTO)

Para mais informações :

www.devmedia.com.br/curso/netmagazine

(21) 3382-5038

DevMedia

Além disso, Os eventos FormCreate, btnCalcularClick, btnSairClick e FormClose fazem uso de uma instância única de TLog que é associada a uma variável interna dos mesmos (log). A obtenção de tal referência é possível invocando-se o método de classe GetInstance que foi definido em TLog. Chamadas ao método RegistrarEvento permitem gravar cada uma das ocorrências relacionadas a estes eventos. Já RegistrarErro torna possível, basicamente, a persistência no arquivo de log de informações sobre exceções durante uma simulação de empréstimo.

Executando a aplicação para testes do padrão Singleton

Quando a execução do projeto TesteSingleton for acionada, será exibida inicialmente a tela apresentada na **Figura 7**. Será necessário, a partir disto, preencher os campos "Valor do Empréstimo" e "%Mensal de Juros", conforme indicado na **Figura 8**.

Clicando na opção "Calcular" será exibido então o valor a ser pago para o empréstimo ao fim de doze meses e na eventualidade de algum erro durante o processo de simulação, uma mensagem será apresentada ao usuário.

Na **Figura 9** está um exemplo do arquivo de log que foi gerado durante os testes aqui descritos.

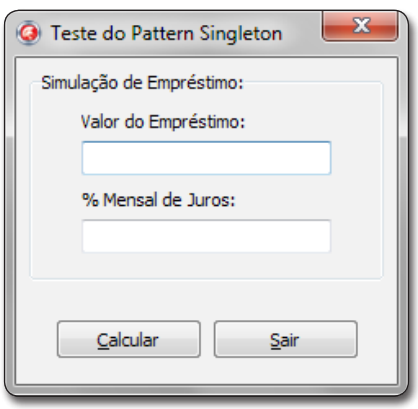


Figura 7. Tela inicial que é exibida ao se executar o projeto TesteSingleton

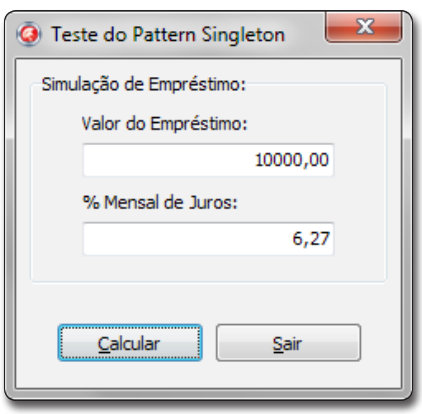


Figura 8. Campos para simulação já preenchidos

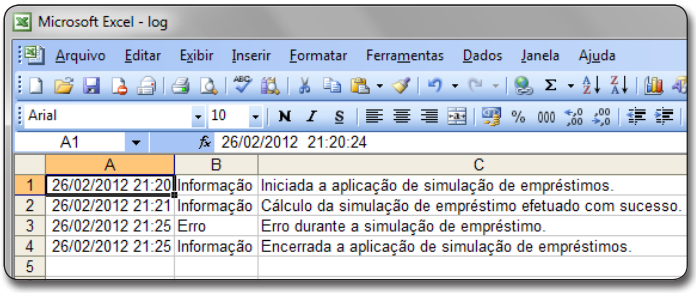


Figura 9. Arquivo log.csv

Conclusão

Conforme discutido aqui e em artigos passados, design patterns são soluções voltadas à solução de problemas com um escopo bem específico dentro do desenvolvimento de software. As recomendações sugeridas por um padrão são fruto de um trabalho que rendeu bons frutos em uma série de projetos anteriores.

O padrão Decorator permite adicionar novos comportamentos a um objeto dinamicamente, sem que isto implique a necessidade de reconstruir toda uma estrutura pré-existente. Logo, conclui-se que o foco deste pattern é apenas estender uma ou mais características, tomando o cuidado de ainda reutilizar outras funcionalidades já concebidas anteriormente e que também atendem a um novo contexto. Já o pattern Singleton estipula a existência de uma única instância para uma classe durante todo o tempo em que uma aplicação permanecer ativa em memória. Este objeto estará disponível através de um ponto de acesso global a tal referência (um método de classe de acesso público). O padrão Singleton pode ser uma boa opção em situações nas quais a construção de várias instâncias de um tipo possa degradar o nível normal de performance de uma operação ou, ainda, levar a efeitos colaterais, com uma provável razão disto sendo o acesso concorrente a um determinado recurso.

A opção por quais patterns empregar em uma solução dependerá sempre de uma análise cuidadosa do que está se procurando atender. Cada projeto possuirá características próprias que devem ser levadas em conta neste processo, sendo uma prática comum que a arquitetura de um sistema combine vários padrões a fim de se cumprirem as metas traçadas inicialmente.

Renato José Groffe
renato.groffe@yahoo.com.br
Atua como consultor em atividades voltadas ao desenvolvimento de softwares há mais de 10 anos. Bacharel em Sistemas de Informação, com especialização em Engenharia de Software. Microsoft Certified Technology Specialist (Web, WCF, Distributed Applications, ADO.NET, Windows Forms), Oracle Certified Associate (PL/SQL), Sun Certified (SCJP, SCWCD), ITIL Foundation V2.

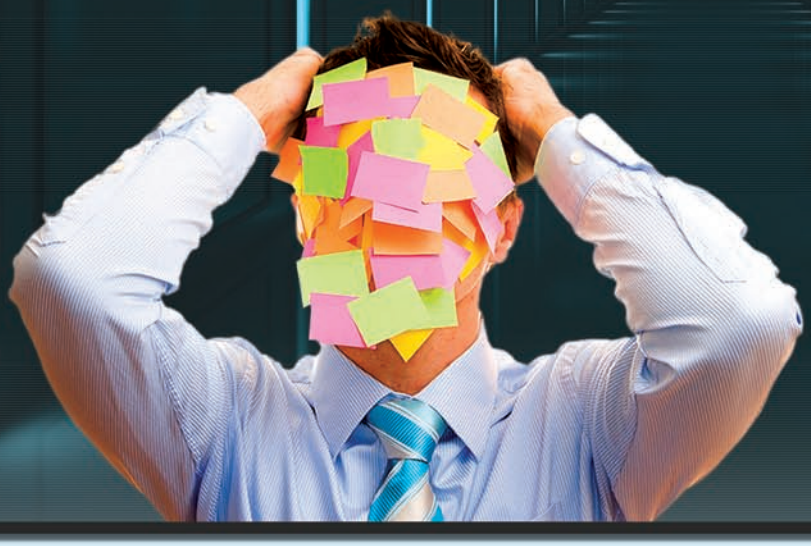
Dê seu feedback sobre esta edição!

A Clubedelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre este artigo, através do link:

www.devmedia.com.br/clubedelphi/feedback

Dificuldades em encontrar conteúdo sobre .Net para iniciantes?



A DevMedia simplifica pra você!

easy .net magazine

Easy .Net Magazine:

A primeira revista digital feita especialmente para quem está dando os primeiros passos em .Net!

Você encontrará exemplos e modelos fáceis e práticos, noções introdutórias e conceitos básicos.

Conheça as vantagens:

- Vídeos e apresentações "embutidos" nas páginas;
- Ilustrações, diagramas e modelos apresentados em PowerPoint;
- Entrevistas em vídeo;
- Códigos-fonte que podem ser copiados com CTRL+C/CTRL+V.

A Easy.Net não é feita em PDF ou Flash!
Ela utiliza um aplicativo 100% online.

DevMedia group

Acesse agora a edição Nº 1 totalmente grátis e comprove

www.devmedia.com.br/easynet

Easy.Net Magazine, seu curso inicial de .Net!

Conhecimento faz diferença!

Faça já sua assinatura digital ! | www.devmedia.com.br/es

Faça um *upgrade* em sua carreira

Em um mercado cada vez mais focado em qualidade, ter conhecimentos aprofundados sobre requisitos, metodologia, análises, testes, entre ouutros, pode ser a diferença entre conquistar ou não uma boa posição profissional.
Sabendo disso a DevMedia lançou uma publicação totalmente especializada em Engenharia de Software.
Todos os meses você pode encontrar artigos sobre Metodologias Ágeis; Metodologias tradicionais (document driven); ALM (application lifecycle); SOA (aplicações orientadas a serviçoes); Análise de sistemas; Modelagem; Métricas; Orientação à Objetos; UML; testes e muito mais. **Assine Já!**

DevMedia